68' MICRO JOURNAL

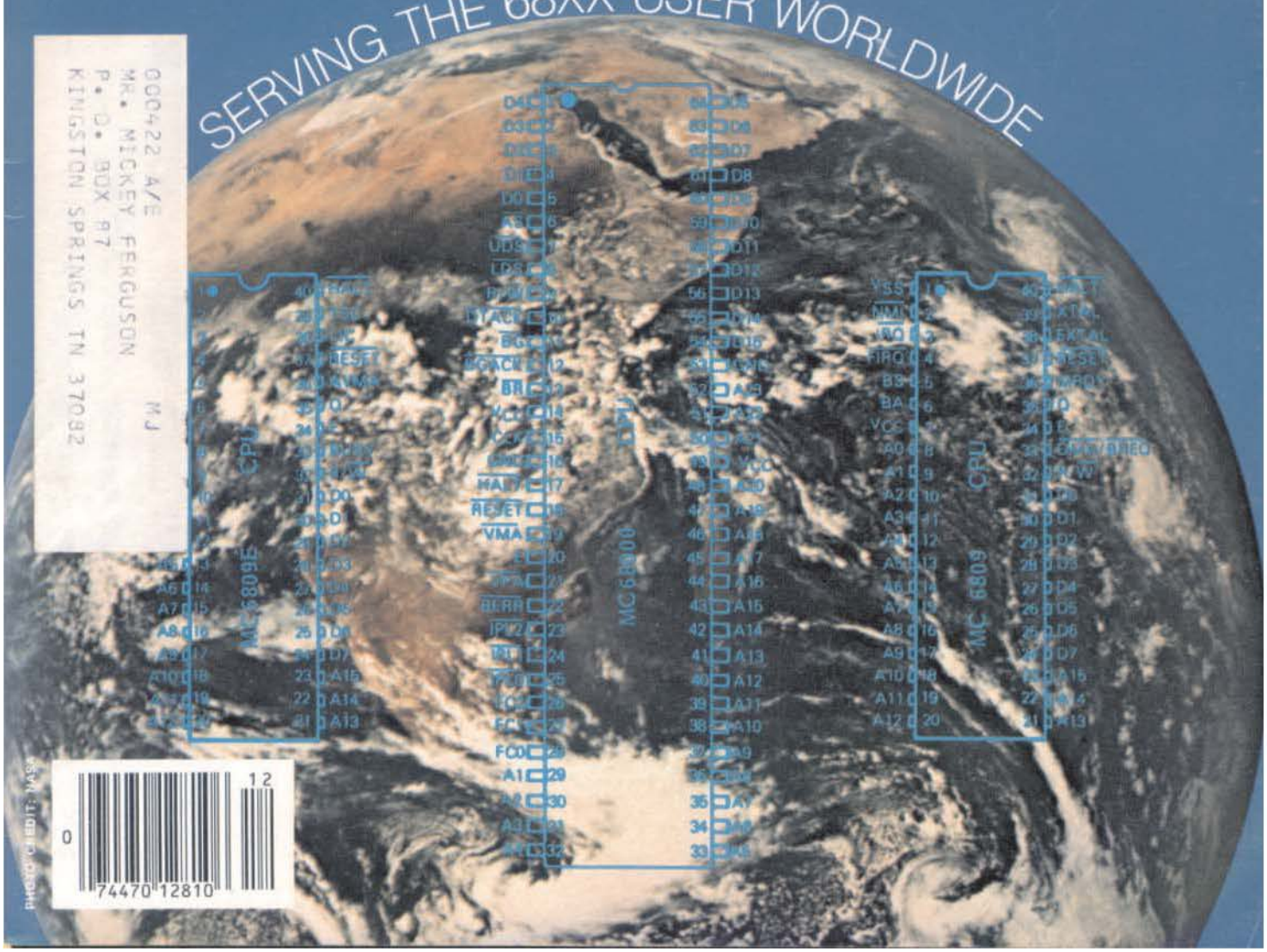
Australia A $4.75 New Zealand NZ $ 6.50
Singapore S $ 9.45 Hong Kong H $23.50
Malaysia M $ 9.45 Sweden 30 SEK

$2.95 USA

VOLUME VII ISSUE XII • Devoted to the 68XX User • December 1985

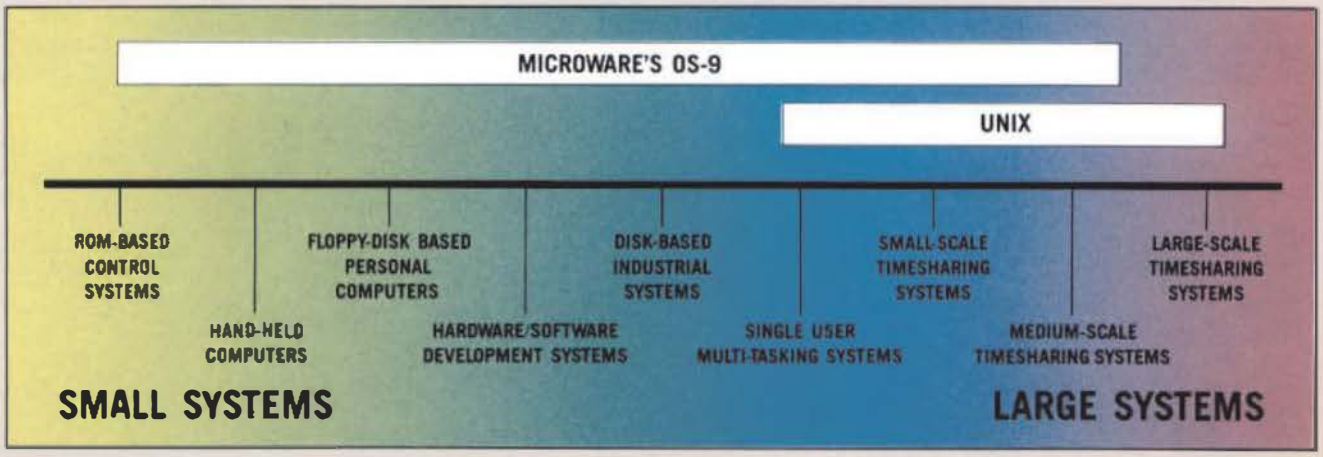
"Small Computers Doing Big Things"

SERVING THE 68XX USER WORLDWIDE



0 74470 12810 12

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTORS

AUSTRALIA
MICROPROCESSOR CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 616-931-2331

ENGLAND
VIVAWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
3-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (28) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER SCANDINAVIAN AB
Sjl Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

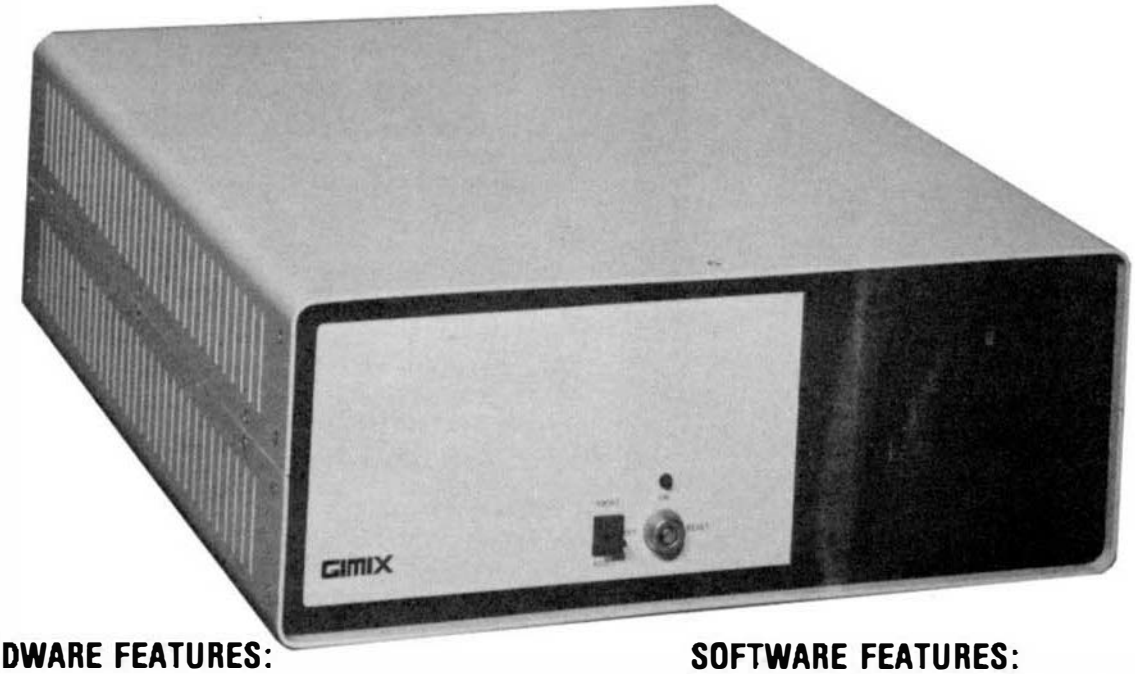
SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL GMBH
Poppstrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 465025

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminals.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-4055

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS – HARDWARE
Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S,9 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

—

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS – WORD PROCESSORS
Technic I Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

—

Stylo Software Inc.
PO Box 916
Idaho Falls, ID 83402
Stylograph - Mail Merge - Spell

Editorial Staff
Don Williams Sr. ..... Publisher
Larry E. Williams ..... Executive Editor
Tom E. Williams ..... Production Editor
Robert L. Nay ..... Technical Editor

Administrative Staff
Mary Robertson ..... Officer Manager
Joyce Williams ..... Subscriptions
Christine Lea ..... Accounting

Contributing Editors

Ron Anderson	Norm Commo
Peter Dibble	William E. Fisher
Dr. Theo Elbert	Carl Mann
Dr. E.M. Pass	Ron Voights
Philip Lucido	Randy Lewis

Special Technical Projects
Clay Abrams K6AEP
Tom Hunt

CONTENTS

Vol.VII, Issue XII December 85

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

Phone (615) 842-4600 or Telex 5106006630

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, TN and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, TN 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add **$9.50** a Year. Other **Foreign Add $12 a Year** for **Surface, Airmail Add $48 a Year.** Must be in **U.S. currency!!**

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be **on either 5 or 8 inch disk** in STYLOGRAPH or TSC Editor format with **3.5 inch column width.** All disks will be returned. Articles submitted **on paper** should be **4.5 inches in width** (including Source Listings) for proper reductions. **Please Use A Dark Ribbon!!** No Blue Ink!!! Single space on 8x11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands **ONLY** should be used: .ap space, .pp paragraph, .fl fill and .nf no fill. Also **please do not format** within the text with **multiple spaces.** We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except **.pg** page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and **must be signed.** Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of **$9.50 for first 20 words** and .45 per word after 20. All classifieds must be **paid in advance.** No classified ads accepted over the phone.

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble
As Published in 68 Micro Journal

The publishers of 68 Micro Journal are proud to announce the publication of Peter Dibbles **OS9 USER NOTES.**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK Typeset -- w/ Source Listings (3-Hole Punched; 8 x 11) **$9.95**

Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk
1 8" SS, SD Disk - - - - $14.95
2 5" SS, DD Disks - - - $24.95

Shipping and Handling; $3.50 per **Book**, $2.50 per **Disk** Set
* All Currency in U.S. Dollars
Foreign Orders Add $4.50 S/H (Surface) or $7.00 S/H (Air Mail)
If paying by check - Please allow 4-6 weeks delivery

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN. 37343

MasterCard

(615) 842-4600
Telex 5106006630

TM - OS9 is a trademark of Microware Systems Corp. and Motorola Inc.
TM - 68' Micro Journal is a trademark of Computer Publishing Inc.

MUSTANG -020 Super SBC™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the **REAL KICK!**"
Only from DATA-COMP

This is the NCC, world beater GMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your specs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

Note: We will include the Motorola 020-BUG at no additional charge. This item alone sells for $500.00. This allows direct coding of 68020 advanced codes. 020-BUG is required for our version of OS9. Please bear in mind, this system is the one others are compared to.

This system is the state-of-the-art star-ship. It runs rings around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be **Turtle** or **Thoroughbred?**

MUSTANG-020 System Prices
Effective November 1985

Mustang-020 SBC, wired & tested with 4 DB25 Serial ports pre-wired, ready to install with your cabinet, P/S, CRT and drives.....$2750.00

M020 Cabinet and P/S, for Mustang-020, less cables..... $269.95

M020 Cables, dual floppy or winchester, specify which - floppy or winchester.....$39.95

M020FC Floppy cabinet and P/S, holds and powers 2 thin-line floppies.....$79.95

M020F Floppy, 80 track, DD/DS.....$269.95

OS-9, SPECIAL Mustang-020 version.....$350.00

MC6881 Floating point co-processor.....Call
(not yet in delivery - Est. Dec. '85)

☛ Call for current winchester prices

Note: for orders of complete systems (Mustang-020, cabinet & P/S, disk drives and OS-9, deduct 5% from total package. (limited time offer)

☛ **Special Winchester Notice** ☚

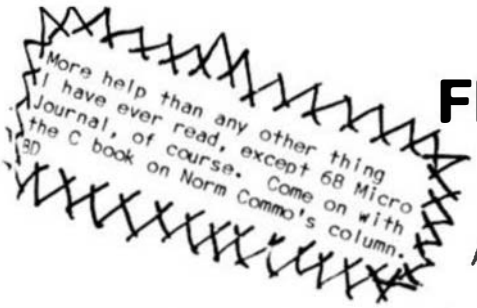
The Mustang-020 device descriptors will allow you to use practically **ANY** winchester drive supported by XEBEC or OMTI controllers.

* * *

Include: $3.50 SBC, cables only S/H. Cabinets include $7.50 S/H.

All checks must be in USA funds. Overseas specify shipping instructions and sufficient funds.

* Mustang-020 is trademark of Data-Comp-CPI



FLEX™ USER NOTES
THE 6800-6809 BOOK

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES,** in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

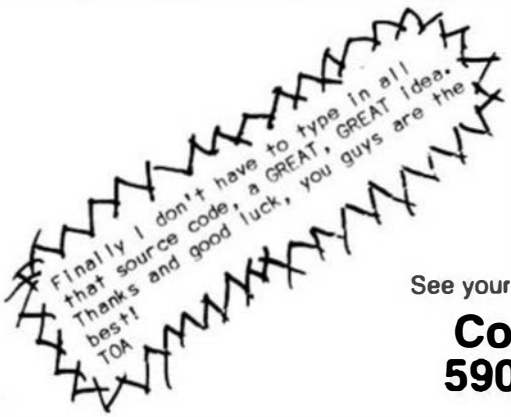
All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC
	(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1,.C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.



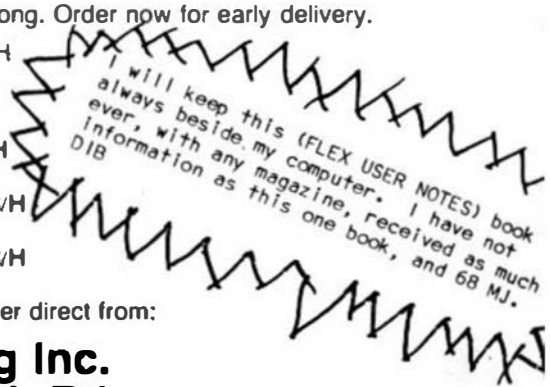
Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H



See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

Languages

Every couple of years in the process of doing this column I have done a column primarily about the various computer languages. It has been a couple of years since the last one so I thought I would again get into the subject (which generally gets me into trouble with some readers). The question always comes up as to why there are so many languages. Each one has its specific area of application where it really "shines". Some languages fit a particular programmer's personality better than others. Some of us are organized to the nth degree and can use a language that requires that we pay very careful attention to detail. Others, like me, are disorganized bumblers that charge right in and start programming. We require a language that forces us to be a bit more organized. With this in mind, let's proceed to the discussion.

BASIC

In my discussion of languages, BASIC always comes first. It was designed to be simple and easy to learn. BASIC does a number of things automatically that are left to the choice of the programmer in other languages. For example, BASIC doesn't require any sort of "format" statement for the output of numbers. PRINT A will get you the value of A on your screen or paper in some "reasonable" format. If A is within a reasonable range, like 10000 or 0.000123, it will be printed in just that way. If it is too small or too big, it will be printed as 1.23E-8 or 7.69503E24. The switch from standard notation to scientific is automatic and requires no instruction from the programmer. Of course most BASIC interpreters have some sort of format statement so that you can print "real" numbers to three or five decimal places, and they have PRINT USING statements adopted straight from COBOL's PIC statement (PICture). These later additions to BASIC are obvious attempts to overcome the limitations. That is not really very important to this discussion. The point is that BASIC frees the beginner from even thinking about such details as number formats. He doesn't even have to worry about whether his numbers are floating point or integer types.

Because of the freedom from detail, BASIC is a very good language to learn as a first computer language. BASIC is usually implemented as an interpreter. That means that it looks directly at your program text and interprets it as it is running. You change a line of program and run it again instantly. That makes it very easy to "debug" a BASIC program. Most all of the BASIC interpreters allow editing of the program without ever leaving the BASIC interpreter. Many offer some fairly sophisticated editing features, though some require you to retype a whole line to remove a comma or add an extra right paren at the end of the line.

BASIC is not without its limitations, however. All of its variables are "GLOBAL". That is, every variable is accessible by every part of the program. The disadvantage of that may not be apparent at this point. Just let me say that LOCAL variables are an aid in keeping the various parts of a large program from interfering with one another .. what the programming theorists call "side effects". BASIC requires a LINE NUMBER on every line of code. It generally doesn't allow you to use meaningful variable names or program labels. GOSUB 1340 doesn't give the reader of the program much clue as to what the subroutine does. In another language that is procedure oriented, the same instruction might be CALCULATE_AVERAGE; which is certainly more informative of the function of the subroutine. Then there is the limitation of variable names being only two characters at most. Some BASICs allow longer variable names but only the first two letters are significant to the interpreter. Consider the following calculation of PAy using HouRs and RaTe.

```
10 IF HR <=40 THEN PA=HR*RT ELSE PA=40*RT+(HR-40)*1.5*RT
```

Of course in BASIC that would be on one line. Now look at the same calculation in Pascal:

```
IF HOURS <= 40 THEN PAY = HOURS * RATE
   ELSE PAY = 40 * RATE +(HOURS-40)*1.5*RATE
```

Because of these limitations, BASIC users don't generate programs that are the most readable to someone else. An unfortunate leftover from the days when microcomputers had very limited memory, is the tendency of some BASIC programmers to run statements all together on one line without any spaces, making them almost impossible for anyone but the original programmer to read.

In an effort to overcome the variable name limitation and the label limitation, many "BASIC pre-compilers" have been written. These take a program in which the programmer has used reasonably long variable names and labels, and converts his text to a form acceptable to the BASIC interpreter. That is, it takes HOURS, PAY, and RATE and converts them to A, B, and C. There are two difficulties with such a pre-compiler. First it adds another step to the use of BASIC. It no longer is a simple interpreter. Change the program and you have to run it through the pre-compiler again. Secondly, the pre-compiler output produces variable names that are even less meaningful than you could assign using two letter names available in BASIC.

If I sound like I am being critical of BASIC, I am, but I use it very frequently. It has GREAT utility in doing little short one time use programs, and in exploring ways of calculating some particular quantity (Algorithm exploration). I use it frequently for what I call exploratory programming. BASIC has very complete functions for handling "strings". A "string" is a string or line of characters to be handled like text, for output to a printer or terminal. Everyone needs a good BASIC interpreter!

I understand that the authors of BASIC (Kemeny and Kurtz) are now actively working on a new standard that has variable names labels, and many of the loop control structures of Pascal (WHILE DO and REPEAT UNTIL). I have not seen any ads for the new BASIC just yet however. These additions will be welcome, and will make BASIC programs vastly more readable.

Because BASIC is an interpreted language, it tends to be slower than some of the higher level languages, and

can be literally 100 times slower than the same program written in assembler. For a complex program with a lot of calculations, there are better ways to go.

Pascal

Pascal's strong points are many. It was written primarily to be a language with which to learn structured programming techniques. Pascal probably more than any other language, forces the programmer to think about what he is doing. It is very strong on the "typing" of variables. You must "declare" each variable before it is used, and you must decide whether the variable is to be of type INTEGER, REAL, CHAR, or BOOLEAN. In addition to those types you can define your own variable types, enumerating the values that they can assume. You can define a TYPE DAYS_OF_WEEK with the values (SUN, MON, TUE, WED, THU, FRI, SAT) and can then create variables of that type.

```
PAY:REAL;
DAY: DAYS_OF_WEEK;
```

The real improvement over BASIC in the other newer languages is the ability to bundle a number of program statements together into one "compound" statement. It is always awkward in BASIC to do multiple things with an IF THEN ELSE. If you need more than one program statement for either the THEN or the ELSE, you have to resort to IF... THEN GOTO..
STATEMENTS.. GOTO. In Pascal it is easy:

```
IF A>B THEN
BEGIN
   B:=B+1;
   K:=K^2;
END
ELSE
BEGIN
   A:= A-1;
   K:=K DIV 2;
END;
```

BEGIN and END are "brackets" that surround a number of program statements that are to be considered one statement. In addition, a statement that won't fit on one line may be split onto two or several lines. the semicolon signals the end of a statement. Actually, the END signals the end of the statement before it, so that statement doesn't require a semicolon, but the semicolon does no harm there, and if it is not there, I always add a statement just before the END and forget to put the semicolon on the previous statement.

Pascal is a very picky language. Because it forces you to declare your intentions explicitly, it catches many errors that would get by a less picky compiler. If you assign the value of a real number to an integer, you have to tell Pascal that you know what you are doing, by using the function ROUND or the function TRUNC. Round rounds the real value to the nearest integer, and TRUNC chops off the fractional part. If E= 2.71828, K:=ROUND(E) would result in K=3. K:=TRUNC(E) would result in K=2. Note that the assignment statement in Pascal requires you to use := (read as becomes equal to). This distinguishes assignments from equality tests as in IF A=B.

I can say from long experience that once a program gets through a Pascal Compiler, it will probably run without creeping, though it may not do precisely what the programmer intended.

All of the Pascal implementations for the 6809 are Compilers as opposed to Interpreters. There are two different kinds of compilers however. One type generates what is called Pseudo Code. The Pseudo code instructions are then interpreted by a simple interpreter. This has the advantage that the same compiler can be used to generate Pseudo code on several different processors, and only the Pseudo code (P-Code) interpreter has to be rewritten to accomodate a different processor. Another advantage of P-Code is that it is generally very efficient. It generates less code than some of the other approaches to a compiler. The other kind of compiler is called a "Native Code" compiler. These generally generate Assembler Source code which must be assembled (generally with a relocatable assembler) and then processed by a "linker - loader". The native code compilers generally generate more code for the same program than the P-code compilers, The native code, however generally runs faster. Native code compilers tend to be multi-pass so that they take considerably longer to compile a program of a given size than the P-Code compilers.

Because Pascal was designed as a teaching tool, and perhaps because it was designed before microcomputers became available, it has no specific file handling procedures. As a result, those who implemented Pascal compilers for Microcomputers each went in a different direction in implementing disk files. This severely limits the "portability" of a Pascal program. Generally all file handling statements have to be rewritten before a program that runs on one compiler will compile on another.

Pascal allows the user to define a data RECORD, a collection of various data types grouped together as an entity. This feature is very valuable in data processing, for setting up data file structures.

Pascal has some very nice qualities. I've tried several times to start to write an article on programming in general trying to use no particular language. I always find that my "pseudo code" is so much like Pascal that I might as well use Pascal for the examples. Pascal programs need few comments. They are very much "self documenting" when written by a programmer who chooses meaningful variable names and procedure names. I strongly recommend that anyone who wants to program in "C" ought to learn Pascal first. More on that below.

"C"

Pardon me if I just use C without the quotes here. C is a language that is similar to Pascal in that it has all the various loop control constructs, but with slightly different syntax. C was primarily designed to be used as a language in which to write system software (operating systems and utilities). Unix is written in C.

Generally C is a little more loose in its syntax than Pascal. It assumes that you know what you are doing. Someone has described C as a "Pascal that is not afraid to get its hands dirty". C is less wordy than Pascal, relying on symbols to a greater extent. For example, BEGIN and END are reduced to the familiar "curly braces" { and }. Pascal uses IF THEN. C uses IF (condition) without the word THEN. That is, the condition is enclosed in parentheses. C lets you print an integer to the terminal or add a number to a character which is something even BASIC won't let you do.

```
10 PRINT CHR$(7)        BASIC
WRITE(CHR(7));          Pascal
putchar(7);               C
```

```
IF CH IN [a..z] THEN CH =CHR(ORD(CH)-32);     Pascal
if (ch>='a' && ch<='z') ch=ch-32;       C
```

The second examples convert lower case a-z to upper case A-Z. Pascal makes you convert the character CH to an integer using the ORD function, subtract the 32 and convert the result back to a CHAR using the CHR function. C lets you subtract 32 from the ch value and doesn't complain. The function IN in Pascal is a nice shorthand way of saying IF CH >= "a" AND CH <= "z". The && in C is the logical AND function.

You can see by these short examples that C is more of a shorthand language that uses more symbols and less words. It lets the programmer who is in the know "cheat" more than Pascal does. It lets the programmer who is not in the know get into trouble much more easily than Pascal does.

Incidentally the second example is hard to show in BASIC because a character as such doesn't exist in BASIC. Characters only exist as part of a string. You'd have to do approximately the following:

```
10 A$= "T"
20 C=ASCII(A$)
30 IF C >= 97 AND C <=123 THEN PRINT CHR$(C-32)
```

Perhaps now you can see why I think (along with many others) that you should learn Pascal before learning C. C is obviously a very useful language. It has more features and capabilities than Pascal, particularly in the area of "pointers", an area beyond the scope of this column. Pointers provide access to any memory address including hardware input output ports. C allows the user to define a data record, a collection of various types of data, called a Structure in C as opposed to the word RECORD in Pascal.

C has one overwhelming virtue. Nearly every implementation of it has followed the standard faithfully. C itself contains no machine dependent features. These are always implemented in a "runtime library" usually written in C. Most implementations follow the standard very closely so that such things as file handling are precisely the same from version to version. That makes programs written in C extremely portable or movable from system to system, even with different processors.

FORTH

Forth is distinguished as a language that is either loved or hated by a given programmer. It's proponents include a lot of very smart people who think it extremely useful and fast to use for program development. Forth comes complete with its own operating system, so that Forth on one computer looks exactly like Forth on another computer. Forth users can with little effort get right down to the hardware level of things. They claim that their programs are smaller and faster than the same program implemented in other languages. I had one Forth user tell me that his Forth programs were smaller in total code than the equivalent program written in Assembler! I dispute both the claim of speed and the claim of smallness. I have not yet seen the standard prime number benchmark run in Forth nearly as fast as C and Pascal implementations of the same program.

There is one undisputed statement that can be made about Forth. A program in Forth is generally more concise (the source text that is) by at least a factor of 2 than any of the languages discussed above. You can without doubt do more with less source code in Forth than most languages (probably APL excluded). Try Forth. If it fits your needs and your personality, by all means go at it.

Special Languages

There are a couple of languages around that were designed particularly to fit the features of the 6809 processor. The two that immediately come to mind are PL/9 and Whimsical. PL/9 follows the model of PL/M, Intel's language written for the 8080. It also follows a language by Tom Crosley called SPL/M written for the 6800 several years ago. These are all low level languages, lower than Pascal or C, at least, but not nearly as low level as Assembler. PL/9 has most of the features of Pascals with the exception of the very rigid data typing. Arithmetic may be done in "mixed mode". That is a calculation that mixes REAL and INTEGER data types may be performed and the result assigned to a REAL variable. The rules for what happens are fairly straightforward, and functions are available for forcing the calculation to proceed in a manner other than the "default". PL/9 is a compiled language, though the 6809 machine code is generated in a single pass. PL/9 is missing a few of the niceties of some of the other languages such as linking in pre compiled modules, though the single pass compiler generally is several times faster compiling the same program (even though it has to compile all the modules every time) than the C or Pascal compilers.

Being somewhat more limited in scope, PL/9 requires a little more of the programmer, but not much. PL/9 generates very efficient output code, usually about half as much as the equivalent C program.

Someone asked me to describe the difference between PL/9 and Whimsical recently. I said that PL/9 was more loosely constructed like C, while Whimsical was rigid like Pascal.

Whimsical is indeed a very fine language. It is VERY much like Pascal. Typing of variables is rigid and type conversion functions must be used to perform mixed mode arithmetic calculations. It generates very efficient code (even more efficient than PL/9) and compiles quickly also. It allows the inclusion of pre-compiled modules in a program, and the use of such modules speeds up the compilation process, though for the equivalent program and no pre-compiled modules, it takes about twice as long as PL/9 to compile. Both languages have excellent support from their authors and suppliers. Both have been around long enough to be just about bug free. Both are designed to take advantage of the specific hardware of the 6809.

Assembler

Originally the only choice for Microprocessor based system users, Assembler is still the choice of many programmers. Some see it as "the only way to go". Since most of you have read my continuing debate here with Dan Farnsworth of Compiler vs Assembler, I won't elaborate further. I recently finished writing a screen editor in PL/9. It was a little slow in some of its functions and I was able to substitute "ASMPROC" procedures for about 2% of the total code and speed it up by a factor of three or four. I simply coded the most repeated loops (places where the program spent most of its time) in Assembler. In many cases speed was of little consequence. If the program can handle my typing on the terminal so that it doesn't miss characters, that is adequately fast. In other cases though, such as searching through the entire file for a word, or going from the bottom of the file to the top, I found myself waiting for the computer. Speeding up the major time consuming loops in those cases made the program run much faster. In order to take advantage of such optimization, of course, you have to know how to program in Assembler. I think you are short changing yourself if you don't get into it a little, at least enough to be able to understand someone else's program.

Assembler is very efficient when the programmer uses it properly. An assembler program is very detailed, very specific to the particular processor, and its source listing is much longer than that for the equivalent program in a higher level language (from five to ten times as long, in fact). For some applications, Assembler is obviously the way to go.

Conclusion

I've obviously used up more than my alloted space this time. Long time readers may want to dig up what I said last time and see if my opinions have changed a little in a few years. Next time we will get back to the regular monthly FLEX feature.

- - -

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

OS-9/68K

Friends, I have made a decision. I'm going to get a 68K system. If I can't afford any better it may be a 68008 system, but it will have at least a half a megabyte of memory, a hard disk drive, and OS-9/68K. If I have to live on peanut butter sandwiches to save the money I'll do it. I'll have to return my borrowed 68008 system soon and going "cold turkey" would probably kill me.

I got a 68008 system on loan a few weeks ago. It came with 512K bytes of memory and two 96tpi floppies. I didn't like it very much at first. I bought a hard disk for my Cimix years ago. Faster disk access sounded good, but what I really wanted was one system disk. My 1.2M Qumes overflowing and I just couldn't get used to swapping floppies like an MSDOS user. OS-9/68K with its standard utilities just barely fits on a DSDD 96tpi disk. There's no room for extras -- certainly not the C compiler. I soon realized how spoiled I have become.

Fortunately Priority One has been selling Shugart hard disk drives for $99. I sent for one with the appropriate cables (one 50-pin socket to socket, one 20-pin socket to card edge, and one 34-pin socket to card edge). I've been talking to Bobby Phillips at Cimix about their 68020 board -- I've been dreaming. He's had nice things to say about the OMTI hard disk controller boards so I tracked an OMTI 5300 down at Arrow Electronics for about $270. I plugged everything together and adjusted the hard disk device descriptor for the OMTI controller and the Shugart drive.

It all worked. It's always a pleasant surprise when that happens. I formatted the new drive and copied everything onto it. Then I started really using the system. Microware has obviously put a lot of work into the 68000. Ease-of-use must have been their watchword. You might think (correctly of course) that they had been using OS-9 for years and knew just what it needed if only they had enough memory to do it. The 68000 gave them enough memory and they did it. Let me give you a few examples.

The shell supports some wild-card matching. A command like:

```
$ del *.r
```

works. It deletes all files with a .r suffix.

The copy command has a -r, for replace, option; when that option is set copy will copy over an existing file if it has to. There is a -w=<directory> option that causes copy to copy a list of files into the named directory. When I wanted to back all my C files up onto a floppy the command:

```
$ copy *.c *.h -w=/d0/C.BACKUP
```

did the trick.

The C compiler has -t=<directory> that lets you specify a directory for C's work files. You put them on the ram disk for a remarkable speedup.

DSave lets you copy only files that are newer than corresponding files in the destination directory. In the assembler the C preprocessor convention of surrounding an included file's name in brackets, <file.h>, applies to the use directive.

```
use <defslist>
```

goes directly to /dd/defs/defslist. But that's not generally required. The new policy is to resolve system dependent names with the linker. You don't include system definitions when you assemble; when you link you include /dd/lib/sys.l which resolves all the system names.

A REAL debugger. It can single-step. Symbols that were declared as globals are available. It disassembles as it goes showing addresses relative to global symbols . I found it quite useful for debugging C programs when they were linked with the -g option (for preserve global symbol information I guess). I bet the debugger alone has saved me at least two hours of debugging time already.

It sounds a bit like I'm in love doesn't it. Well I'm not the romantic type. I always look for the dark cloud attached to the silver lining. One problem is obvious. I've got to give this back. That means that I have to find the money for my own system. Prices range from $1000 up, plus enclosure and peripherals. I'd really like the Cimix, but if they charge the $3000 that I've been guessing as the most it's likely to cost, I don't think I'll be able to afford it. (Of course, if you all buy lots of my books ...)

Another problem: can you imagine the number of 6809 assembly language programs I've written. I like programming of course, but I feel daunted. Converting just the most important ones ... groan!

The hardest part of the dark cloud for me to face is that the software is a little buggy. Nothing serious of course, but the utility commands got my guard down, I was still dazzled by them when I dug into the guts of the system. I was upset when I found that the inside of OS-9/68K has some flaws.

I'll be specific. The Ev$Create option of the the F$Event SVC returns the event ID of the new event in register D0. That's consistent with the other options of the F$Event SVC, but the manual says the I0 will be returned in 0l. The C function chown opens the file who's owner it's changing and leaves it open. This is an easy problem to miss if you don't try to delete a file after you do a chown but before the chown'er terminates.

I've passed over problems like that on my Cimix with a comment or a note in the margin of my manual. On the 68K version they are more upsetting because the finish on the system is generally so smooth.

I suspect that if I found a few bugs in a week of hard work I'll uncover dozens more over the next few months. Judging from my experience so far I'd be surprised if there were any important bugs at the application level. The utility programs run smoothly, and the standard SVCs and C functions all seem fine.

You might want to note the error in the Ev$Create SVC in your manual. I've included a version of chown that works correctly with this column. If you include it on the command line when you compile C programs that use chown it will replace the standard version:

```
cc program.c chown.r ...
```

I got the first system I could find with a 6809 in it. I was one of the first OS-9 Level Two users. I have stayed away from the 68000 mostly because I couldn't afford one, but partly because I've been happy with what I have. Now I'm going to move. Fellow 6809 users, I recommend that you move too. The grass is really greener on the 68K side.

There has been a great deal of discussion about the speed of the 68008. Most people seem to have run benchmarks that indicate that it is slower than the 6809. My subjective opinion is that the 68008 is at least as fast as the 6809. Look at the 68008 as an

8-bit chip. It has lots of big registers and a big address space. Where does this leave the 6809?

I've been hoping that Microware would move all the nice features from OS-9/68K to OS-9. They have a stated policy of keeping the versions of OS-9 compatible as much as possible, but I'd be astonished if they were able to move much of the 68K software to the 6809. Level One systems are already tight on memory. The 68K software would push them way over the edge. Level Two systems could take the load if Microware got clever with address spaces. (Those of you who use IBM mainframes and watched them struggle before they went to XA have a model for the kind of things Microware could do.) I don't think Microware will make that effort. I believe that they will keep Level One and Level Two compatible. The 68K features that they can move they will, but there will be a lot that will remain for the 68K only.

Here's the single-board 68K system configuration I plan on. Note that the single-board isn't specified. I'll fill that in when I know what I can afford.

* Single-board 680xx with at least 512K bytes
* OMTI 5300 SCSI controller
  - Handles a tape drive and two hard disks
* A Hard disk drive
  - At most 40 ms average access time
  - At least 20 Megabytes (I think I'll use the Quantum drive that Priority One is selling for $999)
* A 5.25" DSDD 96tpi floppy drive
* A power supply
  - It looks like this system will need about 10 amps at 5 volts and 5 amps at 12 volts. That's more than I've seen in a single-board computer box. The AT replacement power supplies in Byte look good
* An enclosure
  If I can't find a small enclosure with enough space for a big power supply I'll use one of the AT-alike cabinets.

I don't know of a device driver for a cartridge tape drive on OS-9/68K, so paying the extra money for the OMTI-5300 seems like a bit of a waste. Maybe I'll have time to write a driver for it. If you've ever spent the hours it takes to back a hard disk up to floppy(s!!!), you'll understand why I gambled $70 on the tape support.

I'm back in school now. This column will probably stay pretty short until I get a break.

```
1  #include <stdio.h>
2  #include <modes.h>
3  #include <sgstat.h>
4  #include <direct.h>
5  chown(File, Owner)
6      char *File;
7      int Owner;
8      {
9          int _os_gfd();
10         int _os_sfd();
11         register int path;
12         struct fildes buffer;
13
14         if((path = open(File, S_IREAD+S_IWRITE)) == EOF)
15             return EOF;
16         if(_os_gfd(path, &buffer, sizeof buffer) == EOF){
17             close(path);
18             return EOF;
19         }
20         _strass(buffer.fd_own, ((char *)&Owner) + 2, 2);
21         if(_os_sfd(path, &buffer) == EOF){
22             close(path);
23             return EOF;
24         }
25         close(path);
26         return 0;
27     }
```

68000 User Notes

Philip Lucido
2320 Saratoga Drive
Sharpsville, PA 16150

Hmmmm -- when I last ended this column, I promised the next month's column would return with useful information, after a dreary tirade about lacking anything to write about. The only problem is that column appeared two months ago (assuming I get this thing sent in time for the proper issue - sorry, Don).

OK, obviously I didn't have anything to write about last month. Fact is, I came very close to quitting the column biz, because I lacked the energy to even turn my computer systems on after coming home from work. If I didn't even use the computers this column was ostensibly based upon, how could I keep cranking this thing out?

An answer to that question appeared when I called Don Williams to explain my predicament and try to quit. He wouldn't let me! Instead, he came up with a pretty good idea. While I may not always have something useful to say about OS-9 or the Macintosh, there are (hopefully) a fair number of you out there who would simply like to learn about programming in assembly language on the 68000, a subject which I can certainly blather on about for many months.

So that's where things now stand. Expect to see a tutorial covering 68000 assembly language programming (known as AL from now on) in this space for the time being. I've taught AL before (for the 6809), so with a little luck this will be more than an exercise in filling space, and will prove helpful to many of you out there.

The change of direction has already had a benefit for me. While getting ready to start this column, I was working with some minor test programs using the OS-9 assembler and debugger, when I discovered that symbolic debugging capabilities were now included. For some reason, I was sure that these were planned for the future, but as yet unimplemented. When it turned out I was wrong, I ended up spending the entire evening in front of the terminal, playing with the debugger - the first real recreational use of my computers in many months! Who knows, maybe the burnout can still fade away. (For those of you who don't know what symbolic debugging is, keep reading - it figures heavily in the way I plan to teach AL.)

Beginnings Are Tricky Matters

I need to specify some ground rules here, before I get started. First, it's not easy to know what level of knowledge at which to pitch these columns. For one thing, 68MJ obviously has a very capable readership, and many of you are already familiar with programming in some

AL. Too elementary a tone risks boring you. If I assume too much knowledge, though, I run the opposite risk of losing the very people the tutorial is aimed at. I have no real choice but to take things rather slowly at first. I would certainly be interested in your letters. Please tell me if I'm going too slow or too fast for your tastes.

There are a couple of matters concerning how I cover programming in AL. For one, I must teach the simple mechanics, such as the various opcodes and addressing modes. Equally important, though, is the use of advanced programming techniques made possible by the power of the 68000. Programming in assembly for the 68000 is qualitatively different from programming the various 8 bit processors. This is due to the abundance of registers, addressing modes, and opcodes available. Just knowing what each is capable of is not enough. You should really know something about using all together in an efficient manner. Therefore, expect this column to slip into matters of programming philosophy from time to time.

Finally, I am a firm believer in the learn by doing principle (if it's good enough for me, ...). You can't really hope to get a good feel for AL by reading about it. This means my examples must be things you can try out for yourself. Unfortunately, this forces me to choose a particular operating system to use. The main contenders, being those I have here at home, are OS-9 and the Macintosh OS. While the Mac may be more user friendly, it can also be terribly difficult to program. OS-9 is not without faults as a learning tool, since any program you develop must use the OS-9 module format, and must be position independent, forcing me to introduce some complicated ideas much sooner than I would prefer. Still, OS-9 has that symbolic debugger, which is much nicer to use than the MacsBug debugger. I will therefore be talking about OS-9 most of the time, though I will attempt to say something about using other machines and operating systems so I don't lose too many people.

These are the things you should have to follow along: First, a working assembler and, if at all possible, an assembly language debugger. As I said, I will be using the OS-9 assembler package (made up of r68, 168, and other miscellaneous files), and the OS-9 debugger, **debug**. Next, a 68000 AL reference is most useful. The standard, from Motorola, is **M68000 16/32-Bit Microprocessor Programmer's Reference Manual, part # M68000UM(AD4)**, which is published by Prentice-Hall. I'm not sure how to get it, though I suppose you can try ordering it at any local bookstore (from Prentice-Hall), or perhaps straight from Motorola. Finally, there is a small fan-fold programming reference card, Motorola part number **MC68000(ACI)**, which concisely organizes most of the information you need at hand when actually programming.

Look, Ma - Sixteen Fingers!

Now to start, I have the semi-obligatory explanation of binary and hexadecimal notation. First, consider how we count using the decimal, or base 10, system. Starting with 0, in the one's column, we count up to 1, 2, 3, and so on up to 9. We have now used up all of the single digits, so for the next number, we start the one's column at 0, while the next column to the left, the ten's column, moves from 0 (which we didn't bother showing) to the next digit, 1. That is, 10 follows 9 (easy, huh?). We continue counting by adding 1 to the number in the one's column (that's known as incrementing), until we get to 19. As before, we go back to 0 in the one's column, and increment the value in the ten's column, 20. After we get to 99, we are forced to use yet another column, the hundred's.

Now suppose that instead of ten separate digits 0 to 9, we only have two, 0 and 1. We count in the same way as before, except when a column reaches 1, the next increment will go back to 0, and we have to move to the next column to the left. This is binary, or base 2. Counting in binary, we have 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, corresponding to zero through ten.

So much for counting in sequence, which is obviously very simple. A trickier problem is converting between decimal and binary representations of the same number. No matter what base you represent a number in, each column has a particular "weight" which it adds to the number's value. Thus, in decimal, we have the one, ten, hundred columns. Because of this, a 1 in the hundred's column means a larger number than a 1 in the ten's column. This is known as "positional" notation.

In binary, the columns, from the right side, have weights one, two, four, eight, and so one. Each column has a value exactly twice the value of the column on the immediate right. To convert binary to decimal, we just add up the column weights for all the columns with a 1 in them. Thus, 1010 binary is eight plus two, or 10 decimal. Similarly, 10011001 binary is (decimal) 128+16+8+1, or 153 decimal. To convert decimal to binary, we subtract the largest binary column value possible, corresponding to a 1 in that column, and keep converting, using the number left over. Thus, for 100 decimal, 64 is the largest column, and 100 - 64 = 36. The next column value to use is 32, leaving 36 - 32 = 4, which is itself a column value. Thus, 100 decimal is 1100100 binary.

A note of terminology here. A bit, which you've certainly heard of before, stands for binary digit, so a bit is just one column in a binary number.

In base 16 or hexadecimal notation (hex for short) we have sixteen different single digits. These are 0 to 9, as normal, followed by the "digits" A, B, C, D, E, and F, corresponding to the values ten to fifteen. Don't let the letters throw you. Hex still works like binary and decimal, so the number after F hex is 10 hex, 2A hex follows 29H, 100 hex follows FF hex, etc. In hex, the column values are (decimal) 1, 16, 256, and so on by powers of sixteen.

In converting hex to decimal, we must now multiply a column value by the digit in that column, and add the results, so 123 hex is 1*256 + 2*16 + 3*1 or 291 decimal. Converting decimal to hex involves finding the largest hex column value smaller than the number, finding how many times the column value divides the number, and using that value as the hex digit for that column. The number to convert is reduced by the column value times the hex digit, and the process repeated, until we get to the ones column. For instance, 1000/256 is 3 with remainder 232, 232/16 is 14 (or E hex) with remainder 8, and 8/1 is 8, so 1000 decimal is 3E8 hex.

Now, if you know anything about computers, you've heard that they operate using lots of on-off switches, so it makes sense that they use binary notation, which is just a bunch of on-off switches (bits) lined up in a row. But why, you ask, is hex useful? Well, it obviously takes a lot of bits to represent a sizeable number. For example, 1000 decimal is 1111101000 binary. It turns out that it is very easy to convert between hex and binary, with the hex notation taking far fewer columns than binary. While we know that a computer is really doing things using binary numbers, then, we represent those numbers for human consumption in hex,

To convert hex to binary and binary to hex, you must first know the conversions for the values zero to fifteen:

Decimal	Binary	Hex	Decimal	Binary	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Going hex to binary, just replace each hex digit with the equivalent four bits in binary, so ABC hex is

1010,1011,1100 or 101010111100 binary. From binary to hex, split the number into four bit sections from right, and replace each group of bits with the equivalent hex digit, e.g. binary 1111101000 is 0011,1110,1000 is hex 3E8.

Having gone over base conversions, I'll now reveal that programmers rarely convert by hand. While it is important to understand the principles, there is no reason to put up with the drudgery involved. Instead, programmers typically use calculators like the TI Programmer or the HP-16C. Most debuggers have conversion abilities, also. In OS-9's debug, the 'v' command does conversion duties, printing both the hex and decimal representations of a number. To convert hex 3E8 to decimal, enter the command 'v 3e8'. To convert 1000 decimal to hex, enter 'v #1000'. The pound sign before a number is debug's way of recognizing decimal notation, with hex being the default.

At this point, you might be wondering why, if the computer can convert from hex to decimal and back so easily, why bother with anything other than the familiar decimal? Well, since a computer actually uses binary, there are a number of places where it just makes more sense to stick with hex, which is binary's next-of-kin. For instance, in a 68000, memory addresses go from 0 to 16777215 decimal. This last is not an easy number to remember. In hex, though, the addresses are from 0 to FFFFFF, nice regular numbers which are much easier to handle.

Finally, a convention I'll use from now on. When talking about a particular value, you must be careful to specify what notation is being used. For example, faced with the value 1001, is this binary, decimal, or hex? To avoid confusion, prefixes are used to indicate the base. For binary, the prefix is '%', while for hex it is '$'. Decimal, being the notation we are most used to, takes no prefix. Thus, 1001 = $3E9 = %1111101001. Be careful, though. This convention is not always true. In the OS-9 debugger, a number without a prefix is hex, while a decimal must have a prefix of '#'.

Wasn't That Exciting!

I don't seem to have enough room left to get into the next topic, but like I said before, it's better to start slowly then to rush over the fundamentals. Most everybody out there probably already understands hex and binary, but it wouldn't do to abandon those who don't and need a tutorial like this to understand what will follow.

Anyway, next month I'll actually start demonstrating some actual 68000 AL. Also, unavoidably, I'm going to have to say some things about OS-9 memory modules, relocatable assemblers, and linkers, since that's what I'll be using. I'll also go into the operation of the debugger.

- - -

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

This chapter discusses the new version of the INTROL C compiler for FLEX, UNIFLEX, and OS/9, the new version of the McCosh C compiler for FLEX (marketed by Windrush/S.E. MEDIA - ** see catalog this issue) and covers B-trees, as used in a text search algorithm. It also continues the C-Problems feature skipped in recent columns.

INTROL C COMPILER FLEX UPDATES

The latest version (v.1.6) of the INTROL C compiler contains several improvements and enhancements from earlier versions.

When specifying command line options for the INTROL C compiler modules, a '-' may be substituted for an '=' and a ':' may be substituted for a ';'. This is an attempt to accommodate the peculiarities of the various operating systems under which the INTROL C compiler runs.

A new utility program, called "csorge", is supplied with the new INTROL C compiler. It merges a C source program with its corresponding assembler language source program to form a new assembler language source program containing each C source program statement, followed by the assembler language statements generated from that statement. This is a feature of most other C compilers which was missing from prior versions of the INTROL C compilers. It is especially useful when debugging C programs from the assembler language listing.

A new option is provided to produce a assembler language program listing, which may be placed on the standard output or in a disk file.

The interpretation of the "extern" declaration has been changed to be the same as that used by UNIX. In a multi-module C program, each common variables should be declared with the "extern" declaration in all modules except one, and should appear without the "extern" declaration in exactly one module.

The new preprocessor recognizes two new pseudo-macros, "_LINE_" and "_FILE_", which are automatically replaced by the current line number and current file name, respectively. This is expected to be used in debugging and in error messages. The preprocessor has been modified to check for the correct number of parameters in macro invocations, and to check for several other syntactical situations involving macros.

The compiler was modified to better check for and report several types of errors in the C source program and to properly allocate memory for partially initialized structures. It was also modified to correct several code generation problems, including the improper generation of "bra" when "lbra" should have been generated.

The new linker optionally provides a complete symbol cross-reference output file and renames certain linker output files to avoid naming conflicts with FLEX.

The new loader no longer automatically clears uninitialized data space to zeroes. A command-line option is available to force the clearing of uninitialized data space, if required.

Several compiler bugs were corrected in the new release. One involved the occasional generation of "bra" and similar short relative instructions when "lbra" and similar long relative instructions would have been generated. Another involved the incorrect generation of code involving both the U and X registers which caused the following statement to be processed incorrectly:

```
if ((**ip = ((num >> 4) & 0x0f) + '0') > '9') ...
```

In testing a release version of this new compiler, I was able to verify the new features, but found one existing problem.

The INTROL C compilers do not handle quoted strings or redirection on the command line properly in FLEX. INTROL C once handled both quoted strings and redirection on the command line properly, but does not do so, as of version 1.5. Note that the "I" FLEX command does not offer a complete substitution for input redirection, as FLEX reverts to the terminal after the input file reaches end-of-file, and the user must manually enter control-D.

Contact ** S.E. MEDIA - see catalog this issue if you are interested in their new C compiler.

MCCOSH C COMPILER FLEX UPDATES

The latest version (v.26) of the FLEX version of the McCosh C compiler contains several improvements and enhancements and introduces one new problem. The McCosh Standard C Library and manual are also revised from earlier versions.
The I/O section of the Standard C Library was heavily modified. Random file processing was added, allowing the creation, accessing, seeking, and closing of FLEX random-type files, and allowing the random accessing and seeking of sequential files. The ability to open files to any device attached to the system was incorporated. Terminal access was enhanced by allowing "ttyset" parameters to be altered from within a C program. The lower-level I/O functions, like "open" and "create", have been modified to allow the access to and creation of binary files. The higher-level I/O functions, like "printf", have been changed to buffer data in both directions. Direct access to the FLEX FMS FCB areas is now supported in a manner similar to that used by the INTROL C compiler for FLEX.

The "toupper" and "tolower" functions in the Standard C Library have been revised to check that their arguments are letters (of the appropriate case) before converting them to upper or lower case. K and R is ambiguous on this point, causing a great amount of inconsistency on the handling of these functions. However, the UNIX System V C compiler Standard C Library "toupper" and "tolower" functions check their arguments before converting them.

The "exit" function now places its argument (assumed to be of type "unsigned char" into location $CC20. Thus subsequent programs may check this value and act according to its contents. Since there are versions of FLEX which do not preserve registers across system calls, the compiler now saves all "unused" registers before each system call and restores them afterward.

The McCosh C compiler now recognizes the "unsigned char" type, representing each byte as decimal values 0 to 255, rather than -128 to +127, for the "char" type.

A new section of the manual describes in much greater detail than previously how to generate stand-alone programs written in the C language. It also provides a short monitor (written in C and assembler code) which could be used as-is or as a basis for more sophisticated dedicated use. The monitor is also provided on the release diskette to assist those wanting to use it.

Another new section of the manual provides a set of functions (written by Ron Anderson and published in '68' Micro Journal) which implement the "sine", "cosine", "tangent", "arctangent", "absolute", "square root", "logarithm", "antilogarithm", "polynumial", and "exponential" scientific functions, which are not currently included in the McCosh Standard C Library.

In testing a pre-release version of this new compiler, I was able to verify some of the new features, but found several new problems and discovered that some previously-reported problems still existed. Since this information has already been sent to Windrush, perhaps some of the problems will be solved in the release version of the new compiler.

The "ttyset" parameters are now incorrectly applied to output to the printer and through other precommands, such as "O". The revised McCosh Standard C Library needs to be modified to check for output being sent to other than a terminal device to prevent this problem. To avoid this problem, the user can set the "ttyset" width and depth parameters to zero. In fairness to McCosh, TSC Extended BASIC has this same problem.

The buffering of all I/O except that going through stderr can cause some subtle problems in the order of program output to the standard output device and in the processing of random files. The user having problems with the order of program output to the standard output device may either insert calls on "fflush(stdout)" in appropriate locations, such as before each output to stderr, or may inhibit buffering on stdout with a call on "setbuf(stdout,NULL)" initially. The new random processing functions do not seem to work properly with buffered files, but seem to work correctly with calls on "setbuf(fp,NULL)" immediately after the files are opened.

Programs almost invariably generate larger programs under the new version of the compiler than under the older versions. This is critical only in the case of those programs, such as assemblers, data base managers, and others, which attempt to occupy as much memory as possible in order to improve performance and capabilities, and those which previously barely fit into available memory. Although I have not investigated the matter thoroughly, most of the inflation seems due to the new Standard C Library functions.

The problems related to the linkage of external variables among separately-compiled modules remain. The Microware version of the McCosh C compiler has fixed the majority of these problems.

The command line scanning bug reported earlier remains. Neither the McCosh nor INTROL C compilers handle quoted strings or redirection on the command line properly in FLEX. McCosh has apparently never handled either, although it claims to handle quoted strings on the command line properly. Note that the "I" FLEX command does not offer a complete substitution for input redirection, as FLEX reverts to the terminal after the input file reaches end-of-file, and the user must manually enter control-D.

If a preprocessor command is mis-spelled, the error message may be sent to a temporary file and discarded, sometimes causing weird results or truncated compilations with no seeming explanation. For instance, if "#define" is mis-spelled as "#difine", the compiler will quit with no error message printed on the terminal.

Contact ** S.E. MEDIA - Windrush if you are interested in their new C compiler.

B-TREE SEARCH ALGORITHM

B-Tree structures may be used to search and update large sets of data with great efficiency, using reasonably simple algorithms. They are especially suited for very large files in which higher-level indices may be kept in main memory for speed.

The first published description of B-trees was in 1972 by Bayer and McCreight in "Acta Informatica". Their definition of a B-Tree of order m (as modified by Knuth in "Sorting and Searching") is as follows:

1. Every node except the first has no more than m sons.

2. Every node except for first and final has no less than (2m-1)/3 sons.

3. The first node has at least 2 and no more than 2[(2m-2)/3]+1 sons.

4. All final nodes appear on the same level.

5. All nonfinal nodes with k sons have k-1 keys.

Knuth provides the derivations of the upper and lower bounds for the performance of operations on B-trees, but they are extremely good. In particular, search operations are especially efficient with B-trees.

The insertion (and deletion) algorithms are slower and more complex than the search algorithm, as the nodes may require splitting (and joining). Luckily, many useful applications have no need of a deletion algorithm and perform many more search operations than insertion operations. The related binary search algorithm also has efficient search and slower insertion and deletion operations.

An example of such a B-tree program appears below. It copies non-duplicated lines from its standard input to its standard output, without sorting its input or requiring its input to be in order. It is intended to be used with a speller, after the words have been isolated, to eliminate duplicates before the words are looked up in the dictionary.

Although the B-tree it develops internally is sorted the order of its output follows the order of its input. It could be modified to optionally suppress the output of the non-duplicated lines as they are read and to output the lines from the B-tree when its input has reached end of file. This would slow its use as a filter, as no output would be available while input were still being processed.

```
/* This program gets one line from its
   standard input and searches for the
   line in an evolving binary tree sorted
   alphabetically.  If the line is not found,
   it is inserted in the tree and put to its
   standard output.  If the line is found,
   it is not output.
       Written by Bill Vaughn,
           CVS, U of Rochester, Rochester, NY
*/
#include <stdio.h>
#include <ctype.h>
struct btreenode /* b-tree structure */
{
    char *str;
    struct btreenode *left, *right;
};
struct btreenode *head, *alloc, *hend;
struct btreenode *nextnode(), *allocbtree();
char *catore, *cend; /* char array beginning and end */
char *nextstr(), *alocstr(), *calloc(), *malloc(), *gets();
#define BNODES 4096 /* init alloc for btree nodes */
#define STRING 4096 /* init alloc for strings */
#define ADDNOD 1024 /* Addl alloc for btree nodes */
#define ADDSTR 2048 /* Addl alloc for strings */
main(argc, argv)
char *argv[];
{
    int n;
    char *q;
    struct btreenode *r;
    q = catore = alocstr(STRING); /* string space */
    cend = catore + STRING;
    if (gets(q) == NULL) /* get first line */
        exit(0);
    alloc = head = allocbtree(BNODES); /* tree nodes */
    hend = head + BNODES; /* logic depends upon calloc */
    alloc->str = q;       /* to set all space to NULL */
    puts(q);              /* output the line */
    q = nextstr(q);       /* get new string pointer. */
    while (gets(q) != NULL)
    {
        r = head;
        while (n = strcmp(q, r->str))
        {
            if (n < 0)
            {
                if (r->left != NULL)
                {
                    r = r->left;
                    continue; /* traversing left */
                }
                else
                {
                    /* Creating a left subtree */
                    r->left = nextnode();
                    puts((r->left)->str = q);
                    q = nextstr(q);
                    break; /* get next line */
                }
            }
            else
            {
                if (r->right != NULL)
                {
                    r = r->right;
                    continue; /* traversing right */
                }
                else
                {
                    /* Creating a right subtree */
                    r->right = nextnode();
                    puts((r->right)->str = q);
                    q = nextstr(q);
                    break; /* get next line */
                }
            }
        }
    }
    exit(0);
}
/* Returns pointer to an array of 'n' btree nodes. */
struct btreenode *allocbtree(n)
int n;
{
    struct btreenode *x;
    if (NULL == (x = (struct btreenode *)
        calloc(n, sizeof(struct btreenode))))
    {
        fprintf(stderr, "No memory for tree nodes\n");
        exit(1);
    }
    return (x);
}
/* Returns a pointer to the next available btree node.
   Gets more space if necessary. */
struct btreenode *nextnode()
{
    if (++alloc >= hend)
        hend = (alloc = allocbtree(ADDNOD)) + ADDNOD;
    return (alloc);
}
/* Returns pointer to 'n' bytes for string storage. */
char *alocstr(n)
int n;
{
    char *x;
    if ((x = malloc(n)) == NULL)
    {
        fprintf(stderr, "No memory for stringe\n");
        exit(2);
    }
    return (x);
}
/* Returns a pointer to the next available string space.
   Gets more space if necessary. */
char *nextstr(s)
char *s;
{
    char *x;
    if ((x = s + strlen(s) + 1) >= cend)
        cend = (x = alocstr(ADDSTR)) + ADDSTR;
    return (x);
}
/* Allocates space with sbrk
   (replaces malloc). */
```

```
char *malloc(n)
int n;
{
    char *x;
    x = sbrk(n);
    if ((int)x == -1)
        return NULL;
    else
        return (x);
}
/* Allocates space with sbrk and clears it
   (replaces calloc). */
char *calloc(n,m)
int n,m;
{
    char *x;
    int i;
    x = sbrk(i = n * m);
    if ((int)x == -1)
        return NULL;
    while (--i >= 0)
        x[i] = 0x00;
    return (x);
}
```

C PROBLEM

This program uses the string replace function (described in an earlier chapter) to provide a string replacement program, in which the file names and strings are provided on the command line. The McCosh FLEX command line scanning bug is handled (to the extent possible). The invocation sequence is as follows:

replace old-file new-file old-string new-string

```
#include <stdio.h>
#include <ctype.h>

main(argc,argv)
int argc;
char *argv[];
{
    char line[256], line1[256], *oldstring, *newstring, *x, p = 0;
    FILE *input, *output;
    int count = 0, c, z;

    putc ('\n', stderr);
    if (argc < 4)
    {
        fputs ("use: replace old-file new-file ", stderr);
        fputs ("\"old-string\" \"new-string\"\n", stderr);
        exit (1);
    }
    if (((input = fopen (*++argv, "r")) == NULL) ||
        ((output = fopen (*++argv, "w")) == NULL))
    {
        fputs ("can't open file ", stderr);
        fputs (*argv, stderr);
        exit (1);
    }
    if (*(oldstring = *++argv) == '"') /* McCosh bug */
        for (x = oldstring + 1; *x; ++x)
            *(x - 1) = ((*x == '"') ? '\0' : *x);
    if (*(newstring = *++argv) == '"') /* McCosh bug */
        for (x = newstring + 1; *x; ++x)
            *(x - 1) = ((*x == '"') ? '\0' : *x);
    while (fgets(line, 256, input) != NULL)
    {
        count += strrepl (line1, line, oldstring, newstring);
        fputs (line1, output);
    }
    fclose (input);
    fclose (output);
    for (z = 10000; z; z /= 10)
        if ((c = count / z) || p || (z == 1))
        {
            putc ((c + '0'), stderr);
            count -= c * z;
            ++p;
        }
    fputs (" strings replaced\n", stderr);
}

/*
    strrepl (dst, src, pat, rep) copies "src" to "dst",
    replacing all non-overlapping instances of "pat" by "rep",
    returning the number of replacements performed.
*/
    strrepl (dst, src, pat, rep)
    char *dst, *src, *pat, *rep;
    {
        char x, *p, *t;
        int c = 0;

        while (*src)
        {
            for (p = pat, t = src; (((x = *p++) == *t++) && x); );
            if (x)
                *dst++ = *src++;
            else
                for (p = rep, src = t - 1, ++c; *p; *dst++ = *p++);
        }
        *dst = '\0';
        return (c);
}
```

Extend this program to optionally replace strings representing complete C variable names, rather than partial names. Thus, assuming an old-string of "xxx" and new-string of "yyy", it would change a variable named "xxx" but would not change variables named "zzzxxx" or "xxxzzz" or "zzzxxxzzz", as the original program would.

EXAMPLE C PROGRAM

Following is this month's example C program; it provides an alternative version of the detab program of an earlier chapter. This version expands tabs, rather than replacing them with spaces. The number of columns between tabs may be specified. If it is specified as zero, the file is assumed to be a FLEX formatted file with tabs followed by codes representing compressed spaces. Standard input and output are used, to facilitate the use of the program as a filter on systems supporting pipes and redirection.

```
#include <stdio.h>
#include <ctype.h>

        main(argc, argv)
        int argc;
        char *argv[];
        {
            int col = 1, n = 8, c;
            char *sp;
            FILE *fd, *td;

            fd = stdin;
            td = stdout;
            if (argc > 1)
                n = atoi(argv[1]);
            if (n < 0)
                n = 1;

            while ((c = getc(fd)) != EOF)
            {
                switch (c)
                {
                case '\t':
                    if (n)
                    {
                        do
                            {
                            putc(' ', td);
                            col++;
                            }
                        while ((col % n) != 0);
                    }
                    else
                    {
                        if ((c = getc(fd)) == EOF)
                            exit(0);
                        for (; c; --c)
                        {
                            putc(' ', td);
                            col++;
                        }
                    }
                    break;
                case '\n':
                    putc('\n', td);
                    col = 1;
                case '\0':
                    break;
                case '\r':
                    if (n == 0)
                    {
                        putc('\n', td);
                        col = 1;
                        break;
                    }
                default:
                    putc(c, td);
                    col++;
                }
            }
            exit(0);
        }
```

** All software mentioned is fresh in stock for immediate shipment by S.E. MEDIA, see their catalog - this issue.

UniFLEX User Notes

Kenneth R. Lewis
Automation Engineering
1691 Shelby Oak Dr. N.
Suite 4
Memphis, TN 38134

Last month I began a multi-point discussion of UniFLEX Operating System and various tools, techniques and endeavors. This month, I would like to spend some time on a few of the tools. In particular, "FLEX for UniFLEX" and "tscc" (the newly released "C" compiler from Technical Systems Consultants).

FLEX for UniFLEX is exactly what it's name implies. Most FLEX-hosted software will run in this new environment with no restrictions. The rules are simple....if it uses standard entry points, it will most likely work! Because UniFLEX manages the hardware environment for the user, direct access to ACIA's, PIA's and the like will NOT work! Never the less, having access to the rich set of FLEX utilities and languages, while allowing full use of UniFLEX's facilities is more power than most other systems can ever hope for.

Before anyone gets upset, let me say that I don't claim UniFLEX will replace FLEX or even attempt to. UniFLEX does, however, offer far more in terms of user/productivity enhancement. Most projects, are easier to accomplish in the UniFLEX environment and of course, more people can participate through the same hardware. Use of a common development tool, simultaneously, has distinct advantages over the old "wait your turn" approach! But enough of that, let me tell you about "FLEX for UniFLEX."

FLEX for UniFLEX comes as a standard 8-inch disk with manual and an additional utility-set. These utilities represent an extension to normal FLEX in that they allow user access to the UniFLEX facility that is "hosting" FLEX. As delivered, FLEX for UniFLEX allows up to four "drives" to be attached. The four drives are NOT restricted, though, in that they may be intermixed between floppies and hard-disk drives called "file-disks." A "file-disk" is a specially formatted UniFLEX file that looks like a regular FLEX disk to the hosted system. An advantage here is that any (or all) of the "drives" may reside on one (or more) of the UniFLEX drives and or course, winchester drives are included! Imagine having forty (40) megabytes of disk space divided between your old favorites (from years of work with FLEX) and your newest ventures (developed under UniFLEX).....

I have been using FLEX for UniFLEX for about a year now and have found it to be invaluable as an aid for "transferring" previous development efforts to my new environment. Additionally, I use some of the old standby tools (found only in the FLEX environment) to toy with new ideas or old problems. Once an idea gels, or problem is solved, I move it to UniFLEX through one of the new utilities provided, and continue from there. This article, and many more to follow, are an example of "reversing" the procedure. I write the text (using TSC's newly released "UniFLEX Screen Editor"), check the spelling with Stylo System's "Spelling Checker", text process it with TSC's "pr" and spool it to the printer using the UniFLEX "Enhanced Spooler" package. Once acceptable, I transfer it to a FLEX disk via FLEX for UniFLEX using the "copyuf" utility provided. Larry Williams then transfers the article to a system at '68' micro-journal. Although small, this is just the kind of "convenience" that FLEX for UniFLEX brings to my world.

Another such convenience is TSC's new "tscc" compiler. This new compiler represents a full implementation of Kernighan and Ritchies's "C" language and is NOT lacking in any way for full compliance with "C" for UNIX, System V, Release 1. The compiler is currently available for all standard 68000 family products supported by TSC and for a few custom products. The 68020 version supports Motorola's MC68881 Floating Point Coprocessor by generating "math" instructions directly in-line with the code. Since the MC68020 handles such code as an "exception", there is very little overhead and complex functions are handled directly by the math coprocessor. The new GIMIX 68020 system (advertised in this magazine) utilizes the combination described above and according to an "insider" at TSC, is "....the fastest machine they've ever seen."

The TSC "C" compiler sports a standalone pre-processor (more for the 6809 version than others), a two-pass compiler section, two-pass relocating assembler, an optimizer and a linkage editor. Code can be generated and run with a command a simple as "tscc mycode.c" ! Multiple source files are allowed. Input can be in "C", "assembler", or "relocatable object" format. Conditional compilation is fully supported as well as features that allow for easy generation of libraries. Complex development processes are made simpler through a facility that allows for "selective" re-compilation and linking of modules. Finally, all source code labels are appended to the binary file so that TSC's symbolic debugging tool, "sdb" can show you where you went wrong! A "strip" utility is provided for removing the symbols later.

I've used the "tscc" compiler for six (6) months now and only found one boo-boo in the "macro expansion" feature. This was fixed immediately by TSC and a new copy arrived the next morning! In terms of performance...we here at Automation Engineering have found it to be quite efficient and generates rather compact code. Early on, we discovered that the effort expended trying to find a "sloppy" section of compiler generated code was NOT worth the meager gain in speed or code space. Nowadays, we just crank out code, meet deadlines and smile.....

A "Real World UniFLEX Project"

Speaking of deadlines....a recent project (using the 68010) was made possible and profitable by utilizing "UniFLEX VM" for the 68010 and Versa-Module European (VME) hardware. The combination really shines in the Industrial Control arena. We were asked to develop a system that would sort products packed into six (6) different types of boxes, queue them up (based on boxes per pallet) and route them to another system for palletizing. Each box entered the conveyor system from two (2) feed conveyors. A pneumatic "stop" released the waiting box and allowed it to pass a "LASER" scanning device so that a "barcoded" label could be read by our system (RS-232, 9600 baud, Async., record format w/checksum + acknowledge). Once scanned, the data gathered was "piped" to a data-base program for look-up and verification. The data-base then "piped" a response to the "logic" program (the task that actually managed the conveyor system) and allowed the box to proceed down the conveyor to a "transfer point." Since there were six (6) such transfer points, the boxes were "tracked" by monitoring transitions of photo-eyes placed along the route. At the proper point for a particular box, a

transfer mechanism was activated and the box moved from "main" conveyor to queue. When a queue was filled, and the way clear, all boxes in the queue were released to an "outfeed" conveyor where they arrived (shortly) at a "palletizing" machine. Our system managed all conveyor motors, box "stops" (solenoid actuated, pneumatics), a LASER Scanner, an Operator Keypad (9600 baud), a System Console (ANSI CRT w/Keyboard) and most demanding of all, a bank of five (5) "serial ASCII remote digital interface modules" (used to read all field limit-switches and photocells as well as drive the motors and solenoids).

With all of the above, boxes moving through the system, I/O points being scanned and such. The system was only running at a fraction of it's capacity and all under UniFLEX !!!

Next Month

Next month I will describe another system that is managed by UniFLEX and runs a ROBOT Package Handler....we call it the "One-Armed Octopus"!! Also, I will be describing some wild COLOR Graphics packages that are available for some of the standard VME hardware that runs UniFLEX.

ADA® And The 68000

Theodore F. Elbert
The University of West Florida
Pensacola, Florida 32514

Part 8 Concurrent Processing and Ada's Tasks.

Two of the usual requirements of embedded computer systems are real-time processing and concurrency. These two concepts are related, with real-time requirements often being met by use of concurrent processing, or parallelism. In general, the term concurrent processing refers to those situations in which the execution of more than one sequential thread of code takes place in parallel--or at least it has the appearance of occurring in parallel. The term concurrent programming refers to the tools and techniques for dealing with concurrent processing. Concurrent programming is typical in the development of operating system and real-time system software, and it often distinguishes these kinds of systems from other software systems.

In general, concurrent processing is obtained in one of three ways:

- multicomputing--in which different processes execute on totally different computers, but with detailed communication requirements,

- multiprocessing--in which different processes execute on different processors within the same computer, possibly sharing the same memory and other resources,

-multiprogramming--in which different processes are executed in an interleaved manner on a single processor.

There is currently a trend towards abstract concurrency that is unrelated to the parallelism of the actual target computer. The data flow diagrams found in certain software engineering techniques can be envisioned as networks of concurrent processes, even though eventual implementation has traditionally been in a sequential language. The Ada language supports the concept of abstract concurrency by the inclusion of concurrent programming features as an integral part of the language, regardless of the actual parallelism present in the target environment. Each thread of code in an Ada program is considered to be executing on a single logical processor. A program containing only a single thread of code--the main program, so to speak--executes on a single logical processor, even though the actual implementation may be obtained by multiprocessing or multicomputing. Ada programs executing more than one thread of code in parallel accomplish this concurrency by the use of the Ada task--one task for each additional thread of executing code. Each task also executes on a single logical processor, even though the actual implementation may be obtained through multicomputing, multiprocessing, interleaving on a single processor, or any combination of these techniques. That is, the language and its run-time support environment completely isolate the programmer from the implementation in such a way that the concurrent programming can be developed as though each Ada task were to execute on its own processor. A very specific set of rules determine the task dependencies and the interactions and communications among tasks.

In general, embedded system software applications fall into one of three categories:

- Synchronous or purely cyclic: All tasks performed by the software are, by definition, periodic and execute in a synchronous, sequential pattern according to a fixed schedule. Some very simple control systems fit into this category.

- Mostly cyclic, with some asynchronous events: Most tasks are cyclic in nature and can be scheduled deterministically, but some asynchronous events and burst computing loads must be handled without interfering with the cyclic processing. Many modern weapon control, flight control, and navigation systems fit into this category.

- Asynchronous: Most tasks are asynchronous rather than cyclic in nature. These systems depend almost exclusively on asynchronous, interrupt driven processing. Stimuli arrive at unpredictable times, and software responses must be delivered within rigid time constraints even under peak load conditions. These kinds of systems are said to be event-driven. Command, control, communications, and intelligence systems (C^3I) fall into this category.

The means of achieving the proper scheduling of tasks in embedded computer systems has typically been provided in one of two ways:

- By use of a user written supervisor program implemented in assembly code. This program would normally respond to timed interrupts in order to provide the scheduling process. Context switching--the saving and restoring of processor states--must be provided by this program.

- By use of a real-time operating system kernel or real-time executive. The program must then make operating system calls to access task control, memory management, and interrupt handling facilities of the operating system. Operating system services such as semaphores, mailboxes, and queues are used for synchronization of and communication among concurrently executing tasks. In essence, a real-time kernel provides a basis around which a special-purpose operating system is generated, with the concurrently executing tasks in the role of processes within the operating system.

The use of a real-time operating system to serve as an interface between the hardware and the applications program greatly simplifies the development of embedded system software. In addition, because the operating system itself normally has had wide use, the reliability of the resulting software is usually enhanced.

In the first category of embedded system software--for which synchronous features dominate--the traditional approach to real-time operating systems has been the cyclic executive approach, in which a scheduler allocates to tasks certain time intervals in which to execute. Because different tasks often must execute at different cyclic rates, elaborate methods for allocating processor time must often be developed. The smallest time interval allocated by the scheduler is called a minor cycle. Multiples of the minor cycle--usually powers of two multiples--are called major cycles. The various tasks execute within these cycles, and each task is run to completion and has access to all global data. Asynchronous events, if present, are usually signaled by interrupts. The interrupt service must be provided quickly enough so that it does not interfere with the cyclic portion of the program. Often, the cyclic and asynchronous functions are run as foreground processes, with background processing--that is, processing using whatever processor time is not required by the foreground tasks--performing low-priority functions that are not time critical. A computer self-check program, for example, is often provided as a background task.

For the last two categories of embedded system software--in which asynchronous features either are present to a significant degree, or dominate the software requirements--the cyclic executive approach fails because of the complexity involved. Concurrent tasking is the best approach in this case, and the management of concurrent tasking is most easily obtained by the use of a real-time operating system.

Ada's approach to concurrent programming makes the concurrently executing task a basic program unit. The synchronization of and communications among tasks is provided within the language itself by a process called synchronization by rendezvous. Ada programs, of course, cannot execute without a run-time support environment provided by the implementation. This run-time support environment may well be formed around the same real-time operating system kernel used to provide services to traditional real-time languages. In an Ada program, however, no operating system calls are required--all synchronization of and communications among concurrently executing tasks are provided by features of the language itself. These high order language constructs may cause invocation of operating system calls by the run-time environment, but such action is transparent to the programmer. By providing all task control facilities within the high order language, the portability of the resulting software is greatly enhanced.

The Ada task, together with the subprogram, the package and the generic unit, is one of the four primary program units of the language. Like subprograms and packages, tasks have a two-part form consisting of a specification and a body; both must be present. The specification part defines the interface of the task with other Ada program units, while the body contains the implementation of whatever action the task produces. Unlike the case with subprograms and packages, however, task specifications and task bodies may not be separately compiled, since they are not compilation units. They may be textually separated within a declarative part of a program unit, however, in order to provide desired visibility. In addition, the specification and body may appear in the separately compiled specification and body, respectively, of a package, in which case the task is exported by the package.

While subprogram and package specifications declare the associated entities--that is, they declare subprograms and packages--a task specification declares a task type. Task objects can then be declared in a manner analogous to any other object declaration. The value of an object of a task type designates a task, so that many different tasks of the same task type may be declared.

The specification part of a task defines the interface between the task and other program units. This interface provides the synchronization and communication requirements of the rendezvous mechanism. A rendezvous is produced by the execution of an entry call statement in the calling program unit, coupled with the execution of a corresponding accept statement in the called task. The entry call and the accept statement are interfaced through an entry declaration in the specification part of the called task. An entry declaration may have a formal part that contains a list of formal parameters. Actual parameters in the entry call statement are associated with these formal parameters, thus providing the communications function of the rendezvous. The only kind of declaration permitted in a task specification is an entry declaration. This limitation implies that the only interaction between a task and any other program unit is that obtained through an entry call. That is, since there can be no subprograms or packages declared within the task specification, any subprograms or packages declared within the task body are local to the task body and therefore cannot be accessed by any other program unit.

These concepts can best be illustrated by an example in which a task provides a buffer between two other program units. One of these program units--either the main program or another task--places a character into the buffer by a call to entry PUT, while another program unit retrieves the value by a call to entry TAKE. The task specification has the following form:

```
task BUFFER is
   entry PUT(ELEM : in CHARACTER);
   entry GET(ELEM : out CHARACTER);
end BUFFER;
```

-- This is a task object specification with two entries, PUT and GET. The formal part of each entry contains a single parameter. In the case of entry PUT, the parameter has mode in, so that the parameter value is passed to this task from the calling task. The mode out specified for the parameter in entry TAKE implies that the value is passed from this task to the calling task.

The corresponding task body is:

```
task body BUFFER is
   TEMP : CHARACTER;
```

```
begin
    loop
        accept PUT(ELEM : in CHARACTER) do
            TEMP := ELEM;
        end PUT;
        accept TAKE(ELEM :  out CHARACTER) do
            ELEM := TEMP;
        end TAKE;
    end loop;
end BUFFER;
        -- The task body consists of an endless loop
        containing only the two accept statements
        corresponding to the two entries PUT and
        TAKE.
```

If more than one task object is needed, a task type can be declared by use of the reserved word type:

```
task type BUFFER is
    entry PUT(ELEM  : in CHARACTER);
    entry TAKE(ELEM : out CHARACTER);
end BUFFER;
        -- Here, a task type is declared.  The body
        remains unchanged, but one or more task
        objects must be declared before any task
        is designated.
```

The task object declarations, in their simplest form, might be:

```
BUFFER_1, BUFFER_2, BUFFER_3 : BUFFER;
```

in which case three task objects are declared. Each of these objects designates a separate task, and these tasks will execute concurrently. The task specification and task body are common to each of these tasks. For example, there are six entries now involved:

```
BUFFER_1.PUT    BUFFER_2.PUT    BUFFER_3.PUT
BUFFER_1.TAKE   BUFFER_2.TAKE   BUFFER_3.TAKE.
```

There are also the equivalents of three task bodies.

These particular tasks will execute concurrently with two other tasks. The first of these--the producer task--produces characters that are passed to task BUFFER through the rendezvous mechanism and the entry PUT. The second task--the consumer task--obtains characters from task BUFFER through the rendezvous mechanism and entry TAKE. The semantics of the buffering action is described in the following paragraph.

All three tasks--producer, consumer, and task BUFFER--begin execution simultaneously. Task BUFFER will execute to the first accept statement, at which time it will be suspended awaiting a rendezvous. Meanwhile, the producer and consumer tasks will be executing in parallel. The producer task will eventually execute an entry call statement of the form:

```
BUFFER.PUT(CHAR);
```

at which time the rendezvous will be initiated. The parameter CHAR will be passed to task BUFFER. Upon initiation of the rendezvous, the producer task will be suspended while the accept statement (which ends with end PUT) will be executed. Upon execution of the end PUT statement, the rendezvous is completed, and both the producer task and task BUFFER resume normal execution in parallel. Task BUFFER will execute to the next accept statement and await a rendezvous through entry TAKE. The consumer task will eventually execute an entry call statement of the form:

```
BUFFER.TAKE(CHAR);
```

at which time a second rendezvous will be initiated. The value that was passed to task BUFFER during the first rendezvous is now passed from task BUFFER to the consumer task. When this second rendezvous is completed, task BUFFER completes the loop, again executes the except PUT statement, and then awaits initiation of a rendezvous by the producer task. This process continues, with task BUFFER alternating between rendezvous' with the producer task and rendezvous' with the consumer task.

This example illustrates the general concept of the rendezvous mechanism in an Ada program. The rendezvous provides for both synchronization of and communication between the two tasks involved in the rendezvous. A rendezvous is produced when one task--the calling task--issues an entry call, and the second task--the called task--accepts the call. An entry call is issued by the calling task through the execution of an entry call statement. Note that the entry call statement specifies the entry by name, and that it may include a parameter list by which information is exchanged with the called task. The entry name must be prefixed by the task name.

The called task accepts the entry call when an accept statement with the corresponding entry name is executed. The synchronization of the two tasks is achieved through the specific rules of the rendezvous, while the communication between the tasks, if any, is accomplished by the association of the actual parameters in the entry call statement with the formal parameters in the accept statement. The key point here is that the rendezvous is not effected until the calling task has executed the entry call statement, and the called task has executed a corresponding accept statement.

In a simple rendezvous, task synchronization is produced by the response of the calling task to the entry call statement, and by the response of the called task to the accept statement. These responses are as follows:

- When an executing task encounters an entry call statement, a check is made to determine if the called task is waiting at a corresponding accept statement--that is, an accept statement containing the name of the entry--in which case the rendezvous is effected. If the called task is not waiting at a corresponding accept statement, the execution of the calling task is suspended until a rendezvous can be effected.

- When an executing task encounters an accept statement, a check is made to determine if any task has executed an entry call statement naming the entry associated with the accept statement and is therefore awaiting rendezvous. If there is a calling task awaiting rendezvous, then the rendezvous is effected. If no calling task is awaiting rendezvous, then the execution of the task is suspended until a rendezvous can be effected.

Thus, a rendezvous occurs under one of two conditions:

- A calling task executes an entry call to a called task that is waiting at a corresponding accept statement.

- A called task executes an accept statement for which a calling task is waiting at a corresponding entry call statement.

The simultaneous arrival of the calling task at an entry call, and of the called task at a corresponding accept statement, can be viewed as either case above. Such a situation cannot occur when the underlying hardware has a single processor, of course, because the parallel execution of the two task is only apparent. Even in multiprocessor systems, such simultaneity is very unlikely. The point is that the two tasks are synchronized when the rendezvous is effected. This synchronization is achieved by one of the tasks having its execution suspended until the other is ready for rendezvous.

Basic OS-9

Ron Voigts

One of the earliest commands, if not the first, that you'll learn in OS-9 is BACKUP. One of the first things your user's manual tells you is to BACKUP you're system disk. If you buy a piece of software, it will tell you to make a backup. My first column in the Color Micro Journal emphasized the importance of making backups. I think it only reasonable to devote a column to it.

BACKUP is a rather interesting and unique command. Not all systems have it (although many do). To copy a disk in another system, you might enter a line:

```
COPY A:*.* B:*.*
```

The "*" are wildcards that specify all files. The copy command copies from drive A to drive B everything. Eventually what is on drive A ends up on drive B. Many systems use something like this, but this is not an equivalent of BACKUP, since it is copying file by file.

What differs the OS-9 command, BACKUP, from the wildcard copy routine is that it doesn't concern itself with what is on the disk. The directories, files and modules on the OS-9 disk are totally invisible to BACKUP. When it operates, it makes a "bit image" of the original disk onto the target disk. Sector by sector, byte by byte the the two disk are identical. They are mirror images of each other.

The simplest way to use it, is to enter:

```
OS9:BACKUP
```

This will cause a backup of /D0 to /D1. This is the default condition. If you enter:

```
OS9:BACKUP /D1
```

it will cause a single drive backup on drive /D1 by prompting to insert the source disk and destination disk, alternating until the disk is copied. Finally, you can try:

```
OS9:BACKUP /D0 /D1
```

and the disk in drive /D0 will be copied to /D1. A nice feature about backup is it gives 2 chances to make certain you are really going to do a backup. For the above command it would first ask:

```
Ready to BACKUP /D0 to /D1 ?:
```

This is your first chance. If you really meant to backup /D1 to /D0 or something else enter a N. Otherwise enter a Y and it will continue:

```
XYZ is being scratched
OK ?:
```

XYZ is a dummy name I gave to the disk during formatting. I intended to backup another disk to it. If an important name had come up, instead of the XYZ, I would enter an N. Otherwise, a Y and the backup will start. Unlink copying files, BACKUP is impervious to files' attributes. You may change a files' attributes to protect it from being accidently erased, but when you BACKUP, anything on the the destination disk will be wiped out. So, you get 2 chances, before anything is erased. Another good reason for asking whether you want to BACKUP /D0 to /D1, it gives you time to put a source disk in drive /D0, should you want to backup a disk that doesn't have a commands directory on it. Just remember when it is all over to replace the disk in /D0 with the commands directory.

There are a few things that you can do to effect BACKUP. There three command modifiers you can use. They are:

```
e - exit on read error
s - print single drive prompt
-v - do not verify
```

The e causes BACKUP to abort on any read encountered on the source disk. If you don't use it, the backup will proceed with the errors only being noted to the screen. The s print a single drive prompt. If you specify only one drive in the BACKUP command list, the single drive prompt will be automatically be printed. So, entering:

```
BACKUP /D0
```

or

```
BACKUP S /D0 /D0
```

will do the same thing. The -v will stop the verification routine on the destination disk, which is usually done at the end of backup. I recommend not using these options, unless you have a good reason. For example, use the -v to save some time, just hope that the destination disk is ok. The other option is that you can adjust the amount of memory used. Normally, 4K of memory is dedicated to BACKUP. If you want more you can specify the number of pages or K bytes for it to use. Entering :

```
BACKUP #20K
```

will let it use 20K of memory. You can also use:

```
BACKUP #80
```

to get the same result. Giving it more memory will speed up the process and reduce the amount of switching between drives. If you are a single drive user, you'll especially want to give it more memory. If you don't you'll end up swapping disks maybe 40 times or so. Giving it about 40k will reduce the swapping to perhaps 4 or 5 times.

A CHANGE OF NAMES

There is one drawback, I have found, to BACKUP. Many times you'll want to backup a disk, like the system disk. Then you'll customize it to fit your needs. When finished you've got a disk that no longer is like the parent. The disk is now different, but the name is the same.

To solve the problem, I created DNC, which is short for disk name change. To use it you would enter:

```
dnc <device name>
```

The name can be any mass storage device, such as, /d0 or /d1. If you forget how to use it just enter doc without a device name and a little help list will be printed.

The program opens a path to your device. It goes to LSN 0, the Identification Sector, and reads its name. The old name is printed and a prompt for a new one. If you enter a name larger than 32 characters, it will prompt again for a new name. Just entering a carriage return will cause it to abort, leaving the old name in tack. Two subprograms, I used, are STOC and CTOS. These convert a system string to a C string and a C string to system string, respectively. The difference is, in system strings, the last character has the 8th bit set high. In C strings, the end is marked by a null character ($00). The other thing to the program worth noting is the use of the C file errno.h. It contains the error codes used by exit(). E_BPNAM is bad path name, E_READ is read error, and E_WRITE is write error.

If you use BACKUP to make disks that you customize, then you'll want to use a program like dnc to change the name to reflect the disks identity. Long after the labels has faded or the fallen off, the name will be there to remind you of its purpose. The name can be important.

That's all for now. Have a good month. See you next time.

```
/* DNC.C  -- Disk Name Change       */
/* This program will rename a disk */
/* by Ron Voigts  August 1985       */

#define UPDATE 3
#include <errno.h>
#include <stdio.h>
#include <ctype.h>

main(argc,argv)
int argc;
char *argv[];
```

```
{
   int path, namesize;
   char pname[30];
   char oldname[33], newname[80];
   if (argc == 1)
      help();
   /* set up pathname for open */
   strcpy(pname,argv[1]);
   strncat(pname, "@", 1);

   /* open and read dd sector */
   if ((path=open(pname, UPDATE)) == -1)
      exit(E_BPNAM);
   lseek(path, 311, 0);
   if (read(path, oldname, 32) < 32)
      exit(E_READ);

   /* change old disk name to c format */
   atoc(oldname);

   /* show old name and get new one */
   printf("Old Name:%s\n", oldname);
   namesize=33; /* any number larger than 32 */
   while (namesize >32)
   {
      printf("New Name:");
      gets(newname);
      namesize=strlen(newname);
   }
   if (strlen(newname) == 0)
   {
      printf("No Change Made!");
      exit(0);
   }

   /* convert the new name to system format */
   ctos(newname);

   /* write changed name back */
   lseek(path, 311, 0);
   if (write(path, newname, 32) < 32)
         exit(E_WRITE);
   close(path);
}

atoc(s)
char *s;
{
   int i;
   i=0;
   while (isascii(s[i]))
      i++;
   s[i]=toascii(s[i]);
   s[i+1]='\0';
}

ctos(s)
char *s;
{
   int i;
   i=0;
   while (s[i] != '\0')
      i++;
   s[i-1]|=0x80;
}

help()
{
   printf("Dname <device name>\n");
   printf("   Old Name will be displayed.\n");
   printf("   New Name will be prompted.\n");
   printf("   A <cr> will abort change.\n");
   exit(0);
}
```

Using **K-BASIC**

Frank D. Hottman
LLOYD I/O
19535 NE Glisan
Portland, OR 97230
USA
(503) 666-1097
Telex: 910 380 5448 LLOYD I O

In this episode of the Bit Slicer, we'll examine the use of K-BASIC. For those of you who don't know what K-BASIC is, K-BASIC is a native code BASIC language compiler for FLEX and OS9 systems using the 6809 as the host CPU. As the author of K-BASIC it was felt that I could explain things best. Don Williams has asked me to cover parts of K-BASIC, as space and time permits, over the coming months.

VERSION HISTORY

First of all I want to give a list of the various version that have been released. Over a period of time several versions have been released and to help users identify which version they have and if they need to update their disks. I provide the following list. This list is a reconstruction and may not be totally accurate.

March	1982	- work begins
July	1983	- demonstrated preliminary version at MCC 83
August	1984	- deconstrated preliminary version at Des Moines.
October	1984	- released first versions single pass, OSM assembler no random files no PRINT USING compiled small programs.
January	1985	- double pass, PRINT USING random files, still compiles small programs still uses OSM assembler.
March	1985	- KO assembler replaces OSM assembler, spaces now allowed in expressions, compiles much larger programs, however FLEX version has bug in assembler preventing object code from executing, won't multiply 2*2 and other side effects.
March	1985	- KO assembler bug fixed.
May	1985	- New memory manager added; results in much faster string handling and assignment.
June	1985	- used to develope an integrated accounting system for OS9 level 2 here at LLOYD I/O, beginning work to add single and double precision binary math packages.

July 1985 - multiple statements per line bug fixed. Compiles most any XBASIC program.

August 1985 - version number 1.2:00 assigned and printed on the serial number stickers.

I very strongly suggest that K-BASIC users with versions older than July 8 1985 send in your disks for update. If 60 days has passed since purchase then you need to include the $35 payment, as noted in the manual.

RETURNING TO THE FUTURE

Coming SOON is a binary math package for K-BASIC. This entails a very large investment in development. However, for the benefit derived by a binary math selection (speeds up execution 6-10 times) the small extra expense should pose no problem to those having need for such routines.

The charge for the up coming binary math package will be $50. The price for K-BASIC with the binary math package will be $249. By the way, the package is being contracted from Ron Anderson, and the price increase allows us to compensate Ron for his fine work. This new version of K-BASIC will then include three types of real numbers, however, only one may be used in a compiled program. These are:

Current Type

BCD - defaults to 15 digits, but may be set to any length between 3 and 99 digits. Keep in mind that the longer the mantissa the longer it takes to preform calculations. Useful in accounting applications.

New Binary

SINGLE - about 9 digits of precision but very fast calculations. Useful in many scientific applications.

- And -

DOUBLE - about 17 digits of precision, still very fast. Useful in scientific and accounting applications, but may have round off errors that affect accounting applications.

MAD BEES

Some users have written, phoned, telexed, or by other means communicated that some "features" of K-BASIC are undocumented. As these "suggestions" come in I have made notes and added new sections, notes, and modifications to the manual. By the time you read this article, a new "daisywheel" printed, double sided manual will be available. You may request a new manual with any update done to your disks or a new manual can be purchased for 15 dollars. In the future more additions and notes will be added and a later version of the manual can be purchased.

One problem that seems to be a mystery is during the assembly pass (Ko). When an error is reported such as:

```
Error: UDS
              libs ENDIF.23
```

It means that the label "ENDIF.23" is undefined (UDS = UnDefinedSymbol). What has happened is the compiler (Ka) has failed to generate the label. This is because you have made a programming syntax error: a missing ENDIF statement:

```
              IF his=1
                print hello
ENDIF -------->
              While counter%<9
                print counter%;
                counter%=counter%+2
ENDWHILE ----->
              Loop
                Print "This is the End."
ENDLOOP ------>
```

K-BASIC's compiler makes no report of such missing keywords in these types of structures, so the error is passed down to the assembler where it is reported.

One version of the compiler that was released to only a few users had lost the ability to process more than one statement per line. This error could also cause the above types of errors. Users with this bug should send in their disks for a regular update. The bug resulted from the repair of another bug, but was quickly caught, and exterminated. At last report there has been no reinfestation of any critters into K-BASIC.

XBASIC --> K-BASIC

The manual has a section on translating BASIC programs written for the TSC versions of BASIC known as BASIC, XBASIC, and XPC. K-BASIC will generally compile all of these programs, but the same results may not be obtained. There are several reasons for that.

First, K-BASIC handles expressions differently. There is no type precedence in K-BASIC. All expressions have a type assigned to them. The type may have a default or expected type, but if not, its type becomes that of the first operand in the expression. The remaining operands have to be converted to that type if they are different before any operations are done. For example:

```
BYTE' = WORD% + BYTE2'
  |       |       |
  |       |       +- same type as first
  |       +--------- converted to BYTE type
  +----------------- sets the expression
                      type to byte size
                      integer
```

This expression returns a answer as BYTE size. If XBASIC were capable of the BYTE type, it would have converted the two BYTE sized integers to WORD size and then have done the calculation. If the result were larger than what a BYTE sized variable was able to hold a WORD sized variable would have to be used. This basic difference between XBASIC and K-BASIC, I have found that this doesn't cause a problem in writing new programs, but there are some instances where converting special applications to K-BASIC may not work as expected. One user sent along a quick program to generate banners. It was one of the strangest written programs I have ever seen. It seemed to consist mostly of GOTO's. I never did figure out how it worked, but one of the ideas used was using real numbers in PRINT TAB() statements. The current version of K-BASIC will not automatically convert REAL type operands to any of the integer types.

QUICK DRAW McGAW

K-BASIC is supplied with several example programs. One of them is a prime number generation program. I have tested this program both on XBASIC and K-BASIC. On my 2.25 MHz GIMIX III system the results were:

XBASIC 15 seconds
K-BASIC 7 seconds

This was for finding all primes between 3 and 1000. The program ran in about 9 or 10 seconds under OS9 with no other users on the system. The difference seems to be in the overhead OS9 experiences in multi-tasking.

COMPILE TIME

K-BASIC is supplied with the complete run-time package in source code form, although uncommented. There are about 15,000 lines minimum to process each time a compile is initiated. The run-time assembly code source files have had all comments and extra spaces removed to enhance the speed of assembly by reducing the disk I/O needed to read these files. Even with that there is a lot of disk I/O going on. I have a hard disk on my system, which helps a lot in getting through a compile. I recommend that users employ the fastest step rate on floppy drives.

The following program takes about 5 minutes to compile on my system:

```
PRINT
PRINT "HELLO, WORLD"
PRINT
```

It has been reported to take 15 minutes on 5.25 inch floppy disk systems. The question is: does it take 500 minutes for a 100 line program? The answer is NO. On my system a 1000 (** a program of more than about 500-600 lines, should be a maximum. The 1000 is mentioned as an example, and not recommended **) line program takes about 20 minutes to compile... which would translate to about an hour on a floppy disk system.

This appears to be extremely slow, but if you consider just what is happening it isn't so bad. K-BASIC is NOT an interpreter. Get your program running in BASIC first. Don't use it like one. The best way to work with it is to be very methodical about coding up your applications. A lot of trial and error will consume many hours of your time. If you are using K-BASIC under FLEX, take advantage of the availability the TSC XBASIC interpreter and try out your algorithms.

OS9 users have a problem. XBASIC won't run under OS9. For level two users I am developing a version of the run-time package that is already assembled and is treated as a 6809 subroutine memory module with a table of vectors for the 150 or so subroutines. This should speed up development and reduce the amount of memory used by several different compiled programs. There is no sense in having two copies of the SIN() functions... etc. Also only one copy of the run-time package will be in memory.

AUTOMATION

I have a theory. If I automate most of the operations here at LLOYD I/O, time will be saved that can be used in software development. Well, it is taking time to implement the system. But, progress is being made. I'm using K-BASIC to develope an accounting-invoice system. One of the benefits is faster order processing. I have three printers set up for the various forms. An Okidata 82A prints the 7 inch invoices, an Epson MX-80 prints the mailing label, and another Epson MX-80 prints the serial number stickers. The serial number stickers have eight items of interest:

1. Program name with trademark notice
2. Version number
3. Disk Operating System, and size
4. Date of manufacture
5. Serial number
6. Product number
7. Customer account number
8. Invoice number

My system is set up to allow me to cross reference the serial numbers and invoice numbers to the customer account number. This allows me to process updates easily as long as the sticker is available.

Serial number stickers are placed in the manual, on the disks, and on the customer support registration form. ** SEE NOTE BELOW: (It is our policy to process updates only if the registration form is in our files.) The current form is in the format of a survey. I'll be evaluating these as soon as I have enough to make an accurate poll feasable. On the back of this survey is an area for comments. I always read these comments, looking for suggestions and bug reports. In the future I'll be commenting on these suggestions.

NOTE: For those customers of S. E. MEDIA (CPI) and their dealers, this data is maintained by S.E. MEDIA at their Hixson, Tennessee offices.

- - -

Josef Salbaba

Microcomputer Solutions

PO BOX 4
GAUMBOLDSKIRCHEN
A-2352
AUSTRIA / EUROPE
(0043) 2252/42825

PO BOX 4952
HOLLYWOOD
FLORIDA 33083
USA

South East Media
Mr. Chris Kocher
5900 Cassandra Smith Rd.
Hixson, TN 37343, USA

Austria, 22-Aug-1985

Dear Mr. Kocher:

Thank you for sending back the updated floppies with Kbasic.
We are glad to tell you, that this version works fine, as far as we were able to test compilations.
At about the same date we received a letter from Frank Hoffman offering to exchange diskettes for update.
While you already did the update, we are still missing an update of the manual. F.H. mentioned there exists new manual, he did not send, nor promise to send it.
We consider the delivery of the manual is supposed to be part of the original delivery and herewith we like to request it from your company.

Generally, Kbasic seems to be in good shape, only we have some complaints concerning compilation time efforts. Kbasic needs about 20 minutes to work out a 25-line basic program. Allmost of the time seems to be used up in the assembly with "Ko".
We are using a FT-69 (1MHz 6809 cycle) with 2 disk drives.
Maybe we are doing something wrong ? F.H. will probably have to spend some work on this fact.
We are very interested to follow the futural development of this product and will sign on with the update fee as soon as we have gained some more experience and the delivery were completed with the manual.

Editor's Note: Josef, by the time you read this your manual update should have arrived by air-mail. There were several updates, changes, modifications and improvements over the first 6 months of this year, to K-BASIC. However, it now has stabilized and seems to be ahead of most other software in "clean-up" time.

You cannot imagine the effort that was expanded to make K-BASIC a reasonably "bug free" product in such a short period of time. Your kind remarks as to it's present state of efficiency is appreciated by those engaged in the project.

As to the time it takes to compile. First, actually it takes very little time to compile. For a 50 line XBASIC program the time to compile was less than one minute on a 2 Mhz 6809 FLEX system. The assembly time was longer, 7.35 minutes. The total time from start to a .CMD object file was 8.25 minutes. The resulting object code used about 23 FLEX sectors. The resulting code is fairly efficient. But even 23 total sectors is far less RAM space required than having both XBASIC and the source in memory for normal operation. Considering that XBASIC itself uses 79 sectors, plus your source code. Much better a 23 sector object program - much faster and uses far less RAM. Actually the difference can be "go or no go!"

For a 100 line program, using the same type variables, the total time was only about 3 minutes total time longer, or 11.14 minutes from start to .CMD object code. The time is consumed mostly in the assembly process. K-BASIC is a "virtual memory type" of compiler. That is it uses disk space rather than available RAM space (RAM is very limited). Therefore, you can compile object files restricted only by disk space. Quite a better deal than compilers that run out of available memory.

The trade off is time, which is not too important if you have done your code right (get no errors and the program does what it was designed to do) and the size of your source file. For FLEX users most of the code can be pre-checked in the TSC XBASIC mode, then compiled. For OS9 users it is somewhat more demanding, as there is no BASIC similar to XBASIC running under OS9 at the present. We are working on one and will announce it at soon as we get it market ready. (fingers crossed!)

The other factors that affect compile and assembly time is the type of disk I/O (DMA or serial I/O), the actual type of disk and it's access and read/write speeds. The disk format (interleaving, etc.). System clock speeds, wait states, etc. And of course the disk operating system. All these factors leave a wide swath of value differences. Hence it is difficult to pin down exact compile/assemble times. For some it is much better than for others. But for all, it is certainly much more efficient than BASIC+source. **AND WHERE CAN YOU GET 100, YES, 100 DIGITS OF PRECISION?**

I would be interested in knowing how many of you readers would buy a UniFLEX version of K-BASIC, to compile UniFLEX BASIC? We are looking at it quite seriously, but need some sort of input. Also what about a 68XXX version? **PLEASE** drop me a line and let me know!!

Also please refer to the "Using K-BASIC", this issue, by Frank Hoffman, the author and inventor of K-BASIC.

DMW

- - -

MICROBOX II

William A Broooker
36 Yingally drive
Arana Hills
BRISBANE 4054
QUEENSLAND
AUSTRALIA

I would like share my experiences on the successful construction of a single board computer running the FLEX operating system.

I have been using a 6809 computer at work. It's used as a development system to write and debug programs to download to single board controllers. It has performed well and I have enjoyed using the FLEX operating system.

I decided I would like a FLEX based system at home to do some study on high level languages such as PL9 etc. Because of the cost I decided against a SS50 based system at home.

I started looking at the advertisements for single board computer kits. This would be something I could build up gradually and not break the bank.

After looking in various magazines I found a single board FLEX system which I considered to be the best value for money and having the most innovative features. This was the ** MICROBOX II from MICRO CONCEPTS 8 Skilllcorne Mews, Queens Road, Cheltenham, GL50 2NJ, UK.

The MICROBOX II is based on a MC6809E microprocessor running at 2Mhz. It is equipped with 64K of ram plus another 128K of ram which can be assigned as graphics memory or as a ram disk. It supports two 5.25 DSDD 40 or 80 track disk drives, an eprom disk of 64K as well as the previously mentioned ram disk.

The eprom disk is a small daughter board with four 27128's mounted on it . The utilities disk provided the software to burn the eprom's on board.

The i/o comprises two RS-232 ports, a centronics printer port and an expansion buss. It has a battery backed up real time clock and also has great graphics potential which MICRO CONCEPTS demonstrate on the utilities disk. The MICROBOX II uses the NEC7220 graphics chip for the video generation. There is provision for three modes of displayed text (1) 108*24 chars. (2) 128*72 chars. (3) 84*24 chars. (The 128*72 mode needs a good monitor to do it justice.)

I decided to get the startup kit comprising a double sided p.c board 12" * 9.5", an eprom board, a monitor rom and a utilities disk.

I posted away my overseas bank draft and hoped for the best. In just twelve days I received a note from our Customs Dept. indicating that they were holding a parcel for me. After sorting out the red tape and paying the duty (2%) and sales tax (20%) I was allowed to take the kit home.

The kit was complete and included a stack of documentation and construction notes including an English supplier list for all the parts needed to complete the computer.

Now came the interesting part of tracking down the suppliers of the parts in Australia. (Rather than base their design on a particular family of integrated circuits, MICROCEPTS seem to have selected chips to give the most efficient solution to the function required.) After many phone calls I located all of the parts needed, at the time (early 1985) the NEC7220 and WD1770 disk controller chip were only just being imported into Australia.

I installed all of the ic's into sockets and fitted all the connectors, etc. I had previously decided to mount the board in an Apple lookalike case. I cut the rear section out of the case and fitted an aluminum panel into which I fitted all the i/o connectors. I also made the eprom disk accessible through this panel. An Apple type power supply was used to power the computer.

To test the computer I adapted the Apple keyboard connector to suit the MICROBOX II. The next major expense was a pair of 40 track DSDD disk drives which I mounted into a separate case with a power supply.

After much checking I applied power and was pleasantly surprised to see the MON09 prompt. I borrowed a FLEX disk from work and tried to boot up FLEX. Up came the +++ (magic), full of confidence I tried CAT, up came a catalog of the disk. How about EDIT------ nothing, the cursor disappeared.

Off went a letter to the U.K., back in twelve days was a letter suggesting I should be using TSC FLEX and not SWTPc FLEX. After obtaining TSC FLEX all was well, everything worked perfectly including programs like PL9 and stylograph, well almost, have you tried to find curly brackets on an Apple keyboard.

I replaced the Apple keyboard with a real full ASCII keyboard which completed the construction of the computer. The construction of the kit was a great learning process and I was pleased with the support provided by MICRO CONCEPTS in promptly answering all of my questions.

I believe that the single board computer kits offer enthusiasts with an economical entry into computers based on the FLEX operating system as well as giving the constructor more satisfaction and experience than they would receive by buying a ready made computer.

Hopefully their availability will increase the numbers of FLEX users and give the software writers more incentive to write programs for the 6809.

** Note: a more recent and current address is:

MICRO CONCEPTS
2 St. Stephens Road
Cheltenham, Gloucestershire
GL51 5AA ENGLAND
Tele: (0242) 510525

ISAM

"ISAM"
INDEXED SEQUENTIAL ACCESS METHOD
A File Implementation
for
FLEX9 Operating Systems

By: Joseph D. Condon 8101 Alpine Drive Des Moinee, Iowa 50322 515 278-4581

Continued from p.35 of last month.

```
*******************************************************
*                                                     *
*                 ISAM ADD ROUTINE                    *
*                                                     *
*******************************************************

ADD     TST     OFLAG,U   TEST OPEN FLAG
        LBEQ    ERR155    FILE NOT OPEN
        TST     CFLAG,U   TEST CORRUPT FLAG
        LBNE    ERR150    CORRUPT FILE
        LDD     AFSIZE,U  GET ACTUAL FILE SIZE
        CMPD    MFSIZE,U  COMPARE TO MAX SIZE
        LBHS    ERR188    FILE FULL

        LBSR    MOVVBI    MOVE VARIABLE TO BUFFER 1
        LDD     RECSIZ,U  GET RECORD SIZE
        STD     -2,S      STORE CHAR COUNT
        LEAX    BUFFER,PCR POINT TO BUFFER

ADD1    TST     0,X+      TEST BUFFER CHAR
        BNE     ADD2      NOT NULL RECORD
        LDD     -2,S      GET CHAR COUNT
        SUBD    #1        DECREMENT CHAR COUNT
        STD     -2,S      STORE CHAR COUNT
        BNE     ADD1      TEST NEXT BUFFER CHAR
        LBRA    ERR175    NULL RECORD

ADD2    LDD     AFSIZE,U  GET ACTUAL FILE SIZE
        ADDD    #1        GET RELATIVE RECORD NUMBER
        LBSR    WRREC     WRITE RELATIVE RECORD
        LBNE    EXIT      EXIT WITH ERROR

        LBSR    FIND      FIND LAST RECORD < KEY
        LBNE    EXIT      EXIT WITH ERROR
        LDD     CURREC,U  GET CURRENT RECORD NUMBER
        ADDD    #1        BUMP RECORD NUMBER
        STD     CURREC,U  SET NEW CURRENT RECORD NO

        LDD     AFSIZE,U  GET ACTUAL FILE SIZE
        ADDD    #1        BUMP ACTUAL FILE SIZE
        STD     AFSIZE,U  STORE ACTUAL FILE SIZE
        SUBD    #1        DECREMENT RECORD NUMBER
        LSLB              MULTIPLY RECORD NO BY 2
        ROLA
        LEAX    KEYTAB,U  GET KEY TABLE LOCATION
        STX     -2,S      STORE KEY TABLE LOCATION
        ADDD    -2,S      ADD LOCATION TO D REG
        TFR     D,X       MOVE D REG TO X REG
        LDD     0,X       GET PHYSICAL RECORD NUMBER

        STD     -2,S      SAVE PHYSICAL RECORD NO
        LDD     AFSIZE,U  GET ACTUAL FILE SIZE
        SUBD    CURREC,U  CALCULATE DIFFERENCE
        STD     -4,S      STORE DIFFERENCE

ADD3    LDD     -4,S      GET DIFFERENCE
        BEQ     ADD4      END RECORD INSERT
        SUBD    #1        DECREMENT DIFFERENCE
        STD     -4,S      STORE DIFFERENCE
        LDD     0,--X     GET RECORD POINTER
        STD     2,X       MOVE PNTR TO NEXT LOCATION
        BRA     ADD3      MOVE TO NEXT LOCATION

ADD4    LDD     -2,S      GET NEW RECORD POINTER
        STD     0,X       INSERT NEW POINTER
        LBSR    SETUF     SET UPDATE FLAG
        LBNE    EXIT      EXIT WITH ERROR
        LDD     #0        GET STATUS CODE
        RTS               RETURN

*******************************************************
*                                                     *
*                 ISAM DELETE ROUTINE                 *
*                                                     *
*******************************************************

DELETE  TST     OFLAD,U   TEST OPEN FLAG
        LBEQ    ERR155    FILE NOT OPEN
        TST     CFLAG,U   TEST CORRUPT FLAG
        LBNE    ERR150    CORRUPT FILE
        LDD     CURREC,U  GET CURRENT RECORD NUMBER
        LBEQ    ERR165    START OF FILE
        CMPD    AFSIZE,U  COMPARE TO ACTUAL SIZE
        LBHI    ERR178    END OF FILE
        LDD     RECSIZ,U  GET RECORD SIZE
        LEAY    BUFFER,PCR POINT TO RECORD BUFFER

DEL1    CLR     0,Y+      CLEAR RECORD CHARACTER
        SUBD    #1        DECREMENT RECORD SIZE COUNT
        BNE     DEL1      CLEAR NEXT CHARACTER

        LDD     CURREC,U  DET CURRENT RECORD NUMBER
        LBSR    WRREC     WRITE RELATIVE RECORD
        LBNE    EXIT      EXIT WITH ERROR

        LDD     CURREC,U  GET CURRENT RECORD NUMBER
        SUBD    #1        DECREMENT RECORD NUMBER
        LSLB              MULTIPLY RECORD NO BY 2
        ROLA
        LEAX    KEYTAB,U  DET KEY TABLE LOCATION
        STX     -2,S      STORE BUFFER LOCATION
        ADDD    -2,S      ADD LOCATION TO D REG

        TFR     D,X       MOVE D REG TO X REG
        LDD     0,X       GET PHYSICAL RECORD NUMBER
        STD     -2,S      SAVE PHYSICAL RECORD NO
        LDD     AFSIZE,U  GET ACTUAL FILE SIZE
        SUBD    CURREC,U  SUBT  CT CURRENT RECORD NO
        STD     -4,S      STORE DIFFERENCE

DEL2    LDD     -4,S      GET DIFFERENCE
        BEQ     DEL3      END RECORD INSERT
        SUBD    #1        DECREMENT DIFFERENCE
        STD     -4,S      STORE DIFFERENCE
        LDD     2,X       GET RECORD POINTER
        STD     0,X++     MOVE PNTR TO NEXT LOCATION
        BRA     DEL2      MOVE TO NEXT LOCATION

DEL3    LDD     -2,S      GET NEW RECORD POINTER
        STD     0,X       INSERT NEW POINTER
        LDD     AFSIZE,U  GET ACTUAL FILE SIZE
        SUBD    #1        DECREMENT ACTUAL FILE SIZE
        STD     AFSIZE,U  STORE ACTUAL FILE SIZE
        CMPD    CURREC,U  COMPARE TO CURRENT RECORD
        BHS     DEL4      FINISH DELETE
        ADDD    #1        ADD 1 TO ACTUAL FILE SIZE
        STD     CURREC,U  SET NEW CURRENT RECORD NO

DEL4    LBSR    SETUF     SET UPDATE FLAG
        LBNE    EXIT      EXIT WITH ERROR
        LDD     #0        GET STATUS CODE
        RTS               RETURN

*******************************************************
*                                                     *
*              ISAM REORGANIZE ROUTINE                *
*                                                     *
*******************************************************

REORG   TST     OFLAG,U   TEST OPEN FLAG
        LBEQ    ERR155    FILE NOT OPEN
        LDD     #0        CLEAR D REGISTER
        STD     AFSIZE,U  SET ACTUAL FILE SIZE 0
        STD     RECCNT,PCR SET RECORD COUNTER
        LEAY    KEYTAB,U  POINT TO KEY TABLE

REORG1  STD     0,Y++     STORE KEY VALUE
        ADDD    #1        BUMP KEY VALUE
        CMPD    MFSIZE,U  COMPARE TO MAX FILE SIZE
        BNE     REORG1    STORE NEXT KEY VALUE

REORG2  LDD     RECCNT,PCR GET RECORD COUNTER
        CMPD    MFSIZE,U  COMPARE TO MAX FILE SIZE
        BNE     REORG3    INSERT NEXT KEY
        CLR     CFLAG,U   CLEAR CORRUPT FLAG
```

```
         LDD     #0          POINT TO START OF FILE
         STD     CURREC,U    SET CURRENT RECORD NUMBER
         LBSR    SETUF       SET UPDATE FLAG
         LBNE    EXIT        EXIT WITH ERROR
         LDD     #0          GET STATUS CODE
         RTS                 RETURN

REORG3   ADDD    #1          BUMP RECORD COUNTER
         STD     RECONT,PCR  STORE RECORD COUNTER
         LDD     AFSIZE,U    GET ACTUAL FILE SIZE
         ADDD    #1          BUMP RELATIVE RECORD NO
         LBSR    RRREC       READ RELATIVE RECORD
         LBNE    EXIT        EXIT WITH ERROR
         LEAX    BUFFER,PCR  POINT TO RECORD SUFFER 1
         TFR     X,D         MOVE X REG TO D REG
         ADDD    INPAR2,PCR  ADD MAX RECORD SIZE
         TFR     D,Y         POINT TO RECORD BUFFER 2
         LDD     RECSIZ,U     GET MAX RECORD SIZE
         STD     -2,S        STORE TEMPORARY
         CLR     -3,S        CLEAR TEMPORARY

REORG4   LDA     0,Y+        GET RECORD 2 CHAR
         STA     0,X+        STORE RECORD 1 CHAR
         ORA     -3,S        OR CHAR WITH TEMPORARY
         STA     -3,S        STORE TEMPORARY
         LDD     -2,S        GET CHAR COUNT
         SUBD    #1          DECREMENT CHAR COUNT
         STD     -2,S        STORE CHAR COUNT
         BNE     REORG4      MOVE NEXT BUFFER CHAR
         TST     -3,S        TEST TEMPORARY
         BNE     REORG7      INSERT RECORD POINTER
         LDD     AFSIZE,U    GET ACTUAL FILE SIZE
         LSLB                MULTIPLY BY TWO
         ROLA
         LEAX    KEYTAB,U    POINT TO KEY TABLE
         STX     -2,S        STORE TEMPORARY
         ADDD    -2,S        ADD KEY TABLE LOCATION
         TFR     D,X         MOVE D REG TO X REG
         LDD     0,X         GET CURENT RECORD POINTER
         STD     -2,S        STORE TEMPORARY
         LDD     MFSIZE,U    GET MAX FILE SIZE
         SUBD    AFSIZE,U    SUBTRACT ACTUAL FILE SIZE
         SUBD    #1          ADJUST DIFFERENCE
         STD     -4,S        STORE TEMPORARY

REORG5   LDD     -4,S        GET DIFFERENCE
         BEQ     REORG6      END POINTER SHIFT
         SUBD    #1          DECREMENT DIFFERENCE
         STD     -4,S        STORE TEMPORARY
         LDD     2,X         GET RECORD POINTER
         STD     0,X++       SHIFT POINTER DOWN
         BRA     REORG5      SHIFT NEXT POINTER

REORG6   LDD     -2,S        GET OLD CURRENT POINTER
         STD     0,X         STORE IN KEY TABLE
         LBRA    REORG2      INSERT NEXT RECORD

REORG7   LBSR    FIND        FIND KEY RECORD
         LBNE    EXIT        EXIT WITH ERROR
         LDD     CURREC,U    GET CURRENT RECORD NUMBER
         ADDD    #1          BUMP CURRENT RECORD NUMBER
         STD     CURREC,U    STORE CURRENT RECORD NO
         LDD     AFSIZE,U    GET ACTUAL FILE SIZE
         ADDD    #1          BUMP ACTUAL FILE SIZE
         STD     AFSIZE,U    STORE ACTUAL FILE SIZE
         SUBD    #1          DECREMENT ACTUAL FILE SIZE
         LSLB                MULTIPLY BY TWO
         ROLA
         LEAX    KEYTAB,U    POINT TO KEYY TABLE
         STX     -2,S        STORE TEMPORARY
         ADDD    -2,S        ADD KEY TABLE LOCATION
         TFR     D,X         MOVE D REG TO X REG
         LDQ     0,X         GET CURRENT RECORD POINTER
         STD     -2,S        STORE TEMPORARY
         LDD     AFSIZE,U    GET ACTUAL FILE SIZE
         SUBD    CURREC,U    SUBTRACT CURRENT RECORD NO
         STD     -4,S        STORE TEMPORARY

REORG8   LDD     -4,S        GET POINTER COUNT
         BEQ     REORG9      END POINTER MOVE
         SUBD    #1          DECREMENT POINTER COUNT
         STD     -4,S        STORE POINTER COUNT
         LDD     0,--X       GET RECORD POINTER
         STD     2,X         STORE RECORD POINTER
         BRA     REORG8      MOVE NEXT POINTER

REORG9   LDD     -2,S        GET CURRENT RECORD POINTER
         STD     0,X         STORE IN KEY TABLE
         LBRA    REORG2      INSERT NEXT RECORD POINTER

*****************************************************************

*        GET FILE SPECIFICATION SUBROUTINE

GETFID   LEAX    IFCB,U      POINT TO ISAM FCB
         LDY     CIPNT,PCR POINT TO CALLER FCB
         LEAY    B,Y         POINT CALLER FILE SPEC LOC
         LDD     2,Y         GET FILE SPEC LENGTH
         LDY     0,Y         POINT TO FILE SPEC

         TSTA                TEST FOR LENGTH < 256
         LBNE    ERR110      INVALID FILE SPEC
         LEAX    3,X         POINT TO FCB DRIVE FIELD
         LDA     WDRVNO      GET WORKING DRIVE NUMBER
         STA     0,X+        SET FCB DRIVE DEFAULT
         LDA     #8          GET NAME FIELD LENGTH
GF1      CLR     0,X+        CLEAR NAME AND EXTENSION
         DECA                DECREMENT CHAR COUNT
         BNE     GF1         CLEAR NEXT CHAR
         LDA     #'I         GET ISA CHARACTER
         STA     0,X+        STORE EXTENSION CHAR
         LDA     #'S         GET ISA CHARACTER
         STA     0,X+        STORE EXTENSION CHAR
         LDA     #'A         GET ISA CHARACTER
         STA     0,X+        STORE EXTENSION CHAR
         LEAX    IFCB,U      POINT TO ISAM FCB
         LEAX    3,X         POINT TO DRIVE NO FIELD
         LDA     #8          GET NAME FIELD LENGTH
         STA     COUNT,PCR   SET CHAR COUNT
GF2      TSTB                CHECK FOR END OF STRING
         LBEQ    ERR110      INVALID FILE SPEC
         DECB                DECREMENT STRING LEN COUNT
         LDA     0,Y+        GET STRING CHARACTER
         CMPA    #SPACE      TEST FOR LEADING SPACES
         BEQ     GF2         GET NEXT STRING CHAR
         CMPA    #'3         CHECK FOR DRIVE NO
         BHI     GF3         GET FILE NAME
         CMPA    #'0         CHECK FOR VALID DRIVE NO
         LBLO    ERR110      INVALID FILE SPEC
         SUBA    #$30        CONVERT TO BINARY
         STA     0,X         SET FCB DRIVE NO
         TSTB                TEST FOR END OF STRING
         LBEQ    ERR110      INVALID FILE SPEC
         LDA     0,Y+        GET NEXT STRING CHAR
         CMPA    #'.         TEST FOR SEPARATOR
         LBNE    ERR110      INVALID FILE SPEC
         DECB                DECREMENT STRING LEN COUNT
         LBEQ    ERR110      INVALID FILE SPEC
         LDA     0,Y+        GET NEXT STRING CHAR
         DECB                DECREMENT STRING LEN COUNT
GF3      LEAX    1,X         POINT TO FCB NAME FIELD
         CMPA    #'A         TEST FOR VALID CHAR
         LBLO    ERR110      INVALID FILE SPEC
         CMPA    #'Z         TEST FOR VALID CHAR
         LBHI    ERR110      INVALID FILE SPEC
GF4      STA     0,X+        PUT CHAR IN FCB NAME FIELD
         TSTB                TEST FOR END OF STRING
         LBEQ    OF11        EXIT GETFIO
         DEC     COUNT,PCR   DECREMENT FIELD COUNTER
         BEQ     GF5         END OF FIELD ENCOUNTERED
         LDA     0,Y+        GET NEXT STRING CHAR
         DECB                DECREMENT STRING LEN COUNT
         CMPA    #'.         TEST FOR VALID CHAR
         BEQ     GF6         PUT CHAR IN FCB NAME FIELD
         CMPA    #'-         TEST FOR VALID CHAR
         BEQ     OF4         PUT CHAR IN FCB NAME FIELD
         CMPA    #'_         TEST FOR VALID CHAR
         BEQ     OF4         PUT CHAR IN FCB NAME FIELD
         CMPA    #'0         TEST FOR VALID CHAR
         LBLO    ERR110      INVALID FILE SPEC
         CMPA    #'9         TEST FOR VALID CHAR
         BLE     GF4         PUT CHAR IN FCB NAME FIELD
         CMPA    #'A         TEST FOR VALID CHAR
         LBLO    ERR110      INVALID FILE SPEC
         CMPA    #'Z         TEST FOR VALID CHAR
         LBHI    ERR110      INVALID FILE SPEC
         BRA     OF4         PUT CHAR IN FCB NAME FIELD
GF5      LDA     0,Y+        GET NEXT STRING CHAR
         DECB                DECREMENT STRING LEN COUNT
         CMPA    #'.         TEST FOR SEPARATOR
         LBNE    ERR110      INVALID FILE SPEC
GF6      TSTB                TEST FOR END OF STRING
         LBEQ    ERR110      INVALID FILE SPEC
         TST     COUNT,PCR   CHECK REMAINING NAME CHARS
         BEQ     GF8         END OF NAME FIELD
GF7      LEAX    1,X         BUMP NAME FIELD POINTER
         DEC     COUNT,PCR   DECREMENT COUNTER
         BNE     GF7         BUMP NAME FIELD POINTER
GF8      CLR     0,X+        CLEAR EXT FIELD
         CLR     0,X+        CLEAR EXT FIELD
         CLR     0,X+        CLEAR EXT FIELD
```

```
         LEAX    -3,X        POINT TO START OF EXT FIELD
         LDA     #3          GET EXT FIELD LENGTH
         STA     COUNT,PCR   SET COUNTER
         LDA     0,Y+        GET NEXT STRING CHAR
         DECB                DECREMENT STRING LEN COUNT
         CMPA    #'A         TEST FOR VALID CHAR
         LBLO    ERR110      INVALID FILE SPEC
         CMPA    #'Z         TEST FOR VALID CHAR
         LBHI    ERR110      INVALID FILE SPEC

GF9      STA     0,X+        PUT CHAR IN FCB NAME FIELD
         TSTB                TEST FOR END OF STRING
         BEQ     OF11        EXIT GETF10
         DEC     COUNT,PCR   DECREMENT FIELD COUNTER
         BEQ     OF10        END OF FIELD ENCOUNTERED
         LDA     0,Y+        GET NEXT STRING CHAR
         DECB                DECREMENT STRING LEN COUNT
         CMPA    #'-         TEST FOR VALID CHAR
         BEQ     GF9         PUT CHAR IN FCB NAME FIELD
         CMPA    #'_         TEST FOR VALID CHAR
         BEQ     GF9         PUT CHAR IN FCB NAME FIELD
         CMPA    #'0         TEST FOR VALID CHAR
         LBLO    ERR110      INVALID FILE SPEC
         CMPA    #'9         TEST FOR VALID CHAR
         BLE     GF9         PUT CHAR IN FCB NAME FIELD
         CMPA    #'A         TEST FOR VALID CHAR
         LBLO    ERR110      INVALID FILE SPEC
         CMPA    #'Z         TEST FOR VALID CHAR
         LBHI    ERR110      INVALID FILE SPEC
         BRA     OF9         PUT CHAR IN FCB NAME FIELD

OF10     LDA     0,Y+        GET NEXT STRING CHAR
         CMPA    #$SPACE     TEST FOR SPACES
         LBNE    ERR110      INVALID FILE SPEC
         DECB                DECREMENT STRING LEN COUNT
         BNE     OF10        GET NEXT TRAILING SPACE

OF11     LDD     #0          CLEAR STATUS CODE
         RTS                 RETURN


*        READ RELATIVE RECORD SUBROUTINE

*        D REG = RELATIVE RECORD NUMBER


RRREC    SUBD    #1          ADJUST TO ZERO SIGNIFICANT
         LSLB                SHIFT LEFT B REG
         ROLA                ROTATE LEFT A REG
         LEAY    KEYTAB,U    GET KEY TABLE ADDRESS
         STY     -2,S        STORE KEY TABLE ADDRESS
         ADDD    -2,S        ADD KEY TABLE LOC TO OFFSET
         TFR     D,Y         MOVE D REG TO Y REG
         LDD     0,Y         GET PHYSICAL RECORD NUMBER
         LBSR    POSREC      POSITION TO RECORD
         LBNE    EXIT        EXIT WITH ERROR
         LEAX    IFCB,U      POINT TO ISAM FCB
         LDA     #$11        GET FMS RND READ CODE
         STA     0,X         SET FMS FUNCTION CODE
         LDD     RECSIZ,U    GET RECORD SIZE
         STD     COUNT,PCR   SET CHAR COUNTER
         LEAY    BUFFER,PCR  POINT TO RECORD BUFFER
         STY     -2,S        STORE TEMPORY
         LDD     INPAR2,PCR  GET MAX RECORD SIZE PARAM
         ADDD    -2,S        ADD TO BUFFER LOCATION
         TFR     D,Y         MOVE D REG TO Y REG

RRREC1   LDD     COUNT,PCR   GET CHAR COUNTER
         BNE     RRREC2      MORE CHARACTERS
         LDD     #0          CLEAR STATUS CODE
         RTS                 RETURN

RRREC2   SUBD    #1          DECREMENT CHAR COUNTER
         STD     COUNT,PCR   STORE CHAR COUNTER
         JSR     FMS         READ RELATIVE BYTE
         LBNE    FMSERR      FMS ERROR
         STA     0,Y+        STORE CHAR IN BUFFER
         INC     35,X        BUMP RND CHAR OFFSET
         BNE     RRREC1      READ NEXT CHAR

         LDD     32,X        GET REL RECORD NUMBER
         ADDD    #1          BUMP REL RECORD NUMBER
         STD     32,X        STORE REL RECORD NUMBER
         LDA     #$15        GET FMS POSITION CODE
         STA     0,X         SET FMS FUNCTION CODE
         JSR     FMS         POSITION TO REL RECORD
         LBNE    FMSERR      FMS ERROR
         LDA     #$11        GET FMS GET RND BYTE CODE
         STA     0,X         SET FMS FUNCTION CODE
         LDA     #4          GET SECTOR OFFSET
         STA     35,X        SET RNNO BYTE OFFSET
         BRA     RRREC1      READ NEXT CHAR


*        WRITE RELATIVE RECORD SUBROUTINE

*        D REG = RELATIVE RECORD NUMBER


WRREC    SUBD    #1          ADJUST TO ZERO SIGNIFICANT
         LSLB                SHIFT LEFT B REG
         ROLA                ROTATE LEFT A REG
         LEAY    KEYTAB,U    GET KEY TABLE ADDRESS
         STY     -2,S        STORE KEY TABLE ADDRESS
         ADDD    -2,S        ADD KEY TABLE LOC TO OFFSET
         TFR     D,Y         MOVE D REG TO Y REG
         LDD     0,Y         GET PHYSICAL RECORD NUMBER
         BSR     POSREC      POSITION TO RECORD
         LBNE    EXIT        EXIT WITH ERROR
         LEAX    IFCB,U      POINT TO ISAM FCB
         LDA     #$12        GET FMS RND WRITE CODE
         STA     0,X         SET FMS FUNCTION CODE
         LDD     RECSIZ,U    GET RECORD SIZE
         STD     COUNT,PCR   SET CHAR COUNTER
         LEAY    BUFFER,PCR  POINT TO RECORD BUFFER

WRREC3   LDA     0,Y+        GET CHAR FROM BUFFER
         JSR     FMS         WRITE RELATIVE BYTE
         LBNE    FMSERR      FMS ERROR
         LDD     COUNT,PCR   GET CHAR COUNTER
         SUBD    #1          DECREMENT CHAR COUNTER
         STD     COUNT,PCR   STORE CHAR COUNTER
         BEQ     WRREC4      NO MORE CHARACTERS
         INC     35,X        BUMP RND CHAR OFFSET
         BNE     WRREC3      WRITE NEXT CHAR
         LDD     32,X        GET REL RECORD NUMBER
         ADDD    #1          BUMP REL RECORD NUMBER
         STD     32,X        STORE REL RECORD NUMBER
         LDA     #$15        GET FMS POSITION CODE
         STA     0,X         SET FMS FUNCTION CODE
         JSR     FMS         POSITION TO REL RECORD
         LBNE    FMSERR      FMS ERROR
         LDA     #$12        GET FMS PUT RND BYTE CODE
         STA     0,X         SET FMS FUNCTION CODE
         LDA     #4          GET SECTOR OFFSET
         STA     35,X        SET RNNO BYTE OFFSET
         BRA     WRREC3      WRITE NEXT CHAR

WRREC4   LDD     #0          CLEAR STATUS CODE
         RTS                 RETURN


*        POSITION TO PHYSICAL RECORD SUBROUTINE
*        D REG = PHYSICAL RECORD NUMBER


POSREC   BSR     CSCOFF      CALCULATE OFFSETS
         LBNE    EXIT        EXIT WITH ERROR
         LEAX    IFCB,U      POINT TO ISAM FCB
         LDD     ACCUM4+2,PCR GET SECTOR OFFSET
         STD     32,X        SET FCB CURRENT RECORD NO
         LDA     ACCUM3+3,PCR GET CHARACTER OFFSET
         STA     35,X        SET FCB RANDOM INDEX
         LDA     #$15        GET FMS POSITION CODE
         STA     0,X         SET FCB FUNCTION CODE
         JSR     FMS         CALL FMS
         LBNE    FMSERR      GOTO FMS ERROR
         RTS                 RETURN


*        CALCULATE SECTOR CHARACTER OFFSET

*        D REG = PHYSICAL RECORD NUMBER


CSCOFF   STD     ACCUM1,PCR  STORE IN ACCUMULATOR 1
         LDD     RECSIZ,U    GET RECORD SIZE
         STD     ACCUM2,PCR  STORE IN ACCUMULATOR 2
         LBSR    MUL32       MULT ACCUM1 TIMES ACCUM2
         LDD     ACCUM3,PCR  GET UPPER 16 BITS ACCUM3
         STD     ACCUM4,PCR  STORE IN ACCUMULATOR 4
         LDD     ACCUM3+2,PCR GET LOWER 16 BITS ACCUM3
         STD     ACCUM4+2,PCR STORE IN ACCUMULATOR 4

         LDD     MFSIZE,U    GET MAX FILE SIZE
         STD     ACCUM1,PCR  STORE IN ACCUMULATOR 1
         LDD     #2          GET RECORD POINTER SIZE
         STD     ACCUM2,PCR  STORE IN ACCUMULATOR 2
         LBSR    MUL32       MULT ACCUM1 TIMES ACCUM2

         LBSR    ADD32       ADD ACCUM4 TO ACCUM3

         LDD     #252        GET FIRST SECTOR SIZE
         ADDD    #16         ADD ISAM FILE HEADER LENGTH
         STD     ACCUM4+2,PCR STORE IN ACCUMULATOR 4
         CLR     ACCUM4+1,PCR CLEAR ACCUMULATOR 4
         CLR     ACCUM4,PCR  CLEAR ACCUMULATOR 4

         LBSR    ADD32       ADD ACCUM4 TO ACCUM3
         LBSR    DIV252      DIVIDE ACCUM3 BY 252
```

```
         LDA     #4          SECTOR CHAR FIXED OFFSET
         ADDA    ACCUM3+3,PCR ADD TO CHAR OFFSET
         STA     ACCUM3+3,PCR STORE CHAR OFFSET

         LDD     ACCUM4,PCR GET UPPER 16 BITS ACCUM4
         LBNE    ERR145     SECTOR LIMIT EXCEEDED
         RTS                RETURN

*        MOVE BUFFER 2 TO VARIABLE SUBROUTINE

MOVB2V   PSH     X          SAVE X REGISTER
         LEAX    BUFFER,PCR POINT TO RECORD BUFFER
         STX     -2,S       STORE TEMPORY
         LDD     INPAR2,PCR GET MAX RECORD SIZE PARAM
         ADDD    -2,S       ADD TO BUFFER LOCATION
         TFR     D,X        MOVE D REG TO X REG
         LDY     CIFPNT,PCR GET CALLER IFCB POINTER
         LEAY    20,Y       POINT TO RECORD VAR DESC
         LDD     2,Y        GET VARIABLE LENGTH
         STD     -2,S       STORE VAR LENGTH ON STACK
         LDY     0,Y        POINT TO VARIABLE STRING
         LDD     RECSIZ,U   GET RECORD SIZE
         STD     -4,S       STORE RECORD SIZE ON STACK

MOVBV1   LDD     -2,S       GET VAR COUNT
         BNE     MOVBV2     MORE KEYTAB CHARACTER
         PUL     X          RESTORE X REGISTER
         RTS                RETURN

MOVBV2   SUBD    #1         DECREMENT VAR COUNT
         STD     -2,S       STORE VAR COUNT
         LDD     -4,S       GET RECORD COUNT
         BEQ     MOVBV3     LAST BUFFER CHARACTER
         SUBD    #1         DECREMENT RECORD COUNT
         STD     -4,S       STORE RECORD COUNT
         LDA     0,X+       GET BUFFER CHARACTER
         STA     0,Y+       STORE CHAR IN VARIABLE
         BRA     MOVBV1     MOVE NEXT CHARACTER

MOVBV3   CLR     0,Y+       PUT NULL CHAR IN VAR
         LDD     -2,S       GET VAR COUNT
         BEQ     MOVBV4     LAST NULL CHARACTER
         SUBD    #1         DECREMENT VAR COUNT
         STD     -2,S       STORE VAR COUNT
         BRA     MOVBV3     NEXT NULL CHARACTER

MOVBV4   PUL     X          RESTORE X REGISTER
         RTS                RETURN

*        MOVE VARIABLE TO BUFFER 1 SUBROUTINE

MOVVB1   PSH     X          SAVE X REGISTER
         LEAY    BUFFER,PCR POINT TO BUFFER
         LDX     CIFPNT,PCR GET CALLER IFCB POINTER
         LEAX    20,X       POINT TO RECORD VAR DESC
         LDD     2,X        GET VARIABLE LENGTH
         LDX     0,X        POINT TO VARIABLE STRING
         STD     -2,S       STORE VAR LENGTH ON STACK
         LDD     RECSIZ,U   GET RECORD SIZE
         STD     -4,S       STORE RECORD SIZE ON STACK

MOVVB2   LDD     -4,S       GET RECORD COUNT
         BNE     MOVVB3     MORE VARIABLE CHARACTER
         PUL     X          RESTORE X REGISTER
         RTS                RETURN

MOVVB3   SUBD    #1         DECREMENT RECORD COUNT
         STD     -4,S       STORE RECORD COUNT
         LDD     -2,S       GET VAR COUNT
         BEQ     MOVVB4     LAST VARIABLE CHARACTER
         SUBD    #1         DECREMENT VAR COUNT
         STD     -2,S       STORE VAR COUNT
         LDA     0,X+       GET VARIABLE CHARACTER
         STA     0,Y+       STORE CHAR IN BUFFER
         BRA     MOVVB2     MOVE NEXT CHARACTER

MOVVB4   CLR     0,Y+       PUT NULL CHAR IN BUFFER
         LDD     -4,S       GET RECORD COUNT
         BEQ     MOVVB5     LAST NULL CHARACTER
         SUBD    #1         DECREMENT RECORD COUNT
         STD     -4,S       STORE RECORD COUNT
         BRA     MOVVB4     NEXT NULL CHARACTER

MOVVB5   PUL     X          RESTORE X REGISTER
         RTS                RETURN

*        FIND KEY RECORD SUBROUTINE

FIND     LDD     AFSIZE,U   GET ACTUAL FILE SIZE
         ADDD    #1         BUMP RECORD NUMBER
         STD     ULIMIT,PCR SET UPPER RECORD LIMIT
         LDD     #0         GET STATUS CODE
         STD     LLIMIT,PCR SET LOWER RECORD LIMIT
         STD     CURREC,U   SET CURRENT RECORD NUMBER

FIND1    LDD     LLIMIT,PCR GET LOWER RECORD LIMIT
         ADDD    #1         ADD 1 TO LOWER LIMIT
         CMPD    ULIMIT,PCR COMPARE TO UPPER LIMIT
         BNE     FIND2      CALCULATE NEW CURRENT RECORD
         LDD     LLIMIT,PCR GET LOWER LIMIT
         STD     CURREC,U   SET CURRENT RECORD NUMBER
         LDD     #0         GET STATUS CODE
         RTS                RETURN

FIND2    LDD     LLIMIT,PCR GET LOWER LIMIT
         ADDD    ULIMIT,PCR ADD UPPER LIMIT
         LSRA               DIVIDE BY 2
         RORB
         STD     CURREC,U   SET CURRENT RECORD NUMBER
         LBSR    RRREC      READ RELATIVE RECORD
         LBNE    EXIT       EXIT WITH ERROR
         LEAX    BUFFER,PCR POINT TO BUFFER
         STX     -2,S       STORE TEMPORY
         LDD     INPAR2,PCR GET MAX RECORD SIZE
         ADDD    -2,S       ADD BUFFER 1 LOC
         TFR     D,X        MOVE D REG TO X REG
         LEAY    BUFFER,PCR POINT TO BUFFER
         LDD     RECSIZ,U   GET RECORD SIZE
         STD     -2,S       STORE RECORD SIZE ON STACK

FIND3    LDD     -2,S       GET RECORD COUNT
         BEQ     FIND4      GREATER OR EQUAL
         SUBD    #1         DECREMENT RECORD COUNT
         STD     -2,S       STORE RECORD COUNT
         LDA     0,X+       GET BUFFER 1 CHARACTER
         CMPA    0,Y+       COMPARE TO BUFFER 2 CHAR
         BLO     FIND5      LOWER
         BHI     FIND4      GREATER OR EQUAL
         BRA     FIND3      COMPARE NEXT CHAR

FIND4    LDD     CURREC,U   GET CURRENT RECORD NUMBER
         STD     ULIMIT,PCR SET NEW UPPER LIMIT
         BRA     FIND1      COMPARE AGAIN

FIND5    LDD     CURREC,U   GET CURRENT RECORD NUMBER
         STD     LLIMIT,PCR SET NEW LOWER LIMIT
         BRA     FIND1      COMPARE AGAIN

*        SET UPDATE FLAG AND MARK FILE CORRUPT

SETUF    TST     UFLAG,U    TEST UPDATE FLAG
         BEQ     SETUF1     SET FLAG AND MARK FILE
         LDD     #0         GET STATUS CODE
         RTS                RETURN

SETUF1   INC     UFLAG,U    SET TO NON ZERO
         LEAX    IFCB,U     POINT TO ISAM FCB
         LDD     #1         GET FIRST RELATIVE REC NO
         STD     32,X       SET FCB REL RECORD
         LDA     #$15       GET FMS POSITION CODE
         STA     0,X        SET FCB FUNCTION CODE

         JSR     FMS        POSITION TO RECORD 1
         LBNE    FMSERR     GOTO FMS ERROR
         LDA     #$12       GET FMS PUT RANDOM CODE
         STA     0,X        SET FMS FUNCTION CODE
         LDA     #12        GET BYTE OFFSET
         STA     35,X       SET FCB BYTE OFFSET
         LDA     #$FF       GET NON ZERO CODE
         JSR     FMS        PUT RANDOM BYTE
         LBNE    FMSERR     GOTO FMS ERROR
         INC     35,X       BUMP FCB BYTE OFFSET
         LDA     #$FF       GET NON ZERO CODE
         JSR     FMS        PUT RANDOM BYTE
         LBNE    FMSERR     GOTO FMS ERROR
         LDD     #0         GET STATUS CODE
         RTS                RETURN

*        ADD ACCUM3 + ACCUM4 SUBROUTINE

*        ACCUM3 = SUM

ADD32    LDA     ACCUM3+3,PCR GET ACCUM 3
         ADDA    ACCUM4+3,PCR ADD TO ACCUM 4
         STA     ACCUM3+3,PCR STORE IN ACCUM 3
         LDA     ACCUM3+2,PCR GET ACCUM 3
         ADCA    ACCUM4+2,PCR ADD TO ACCUM 4
         STA     ACCUM3+2,PCR STORE IN ACCUM 3
         LDA     ACCUM3+1,PCR GET ACCUM 3
         ADCA    ACCUM4+1,PCR ADD TO ACCUM 4
         STA     ACCUM3+1,PCR STORE IN ACCUM 3
```

```
         LDA     ACCUM3,PCR GET ACCUM 3
         ADCA    ACCUM4,PCR ADD TO ACCUM 4
         STA     ACCUM3,PCR STORE IN ACCUM 3
         RTS                RETURN


*        MULTIPLY ACCUM1 ACCUM2 SUBROUTINE

*        ACCUM3 = PRODUCT

MUL32    CLR     ACCUM3,PCR CLEAR FIRST BYTE OF ACCUM3
         CLR     ACCUM3+1,PCR CLEAR 2ND BYTE OF ACCUM3
         LDA     ACCUM2+1,PCR GET LOWER ACCUM2
         LDB     ACCUM1+1,PCR GET LOWER ACCUM1
         MUL              MULTIPLY A REG TIMES B REG
         STD     ACCUM3+2,PCR SAVE RESULTS
         LDA     ACCUM2+1,PCR GET LOWER ACCUM2
         LDB     ACCUM1,PCR GET UPPER ACCUM1
         MUL              MULTIPLY A REG TIMES B REG
         ADDD    ACCUM3+1,PCR ADD ACCUM3 TO RESULT
         STD     ACCUM3+1,PCR STORE IN ACCUM3
         BCC     MUL321     NO CARRY

         INC     ACCUM3,PCR ADD CARRY TO ACCUM3

MUL321   LDA     ACCUM2,PCR GET UPPER ACCUM2
         LDB     ACCUM1+1,PCR GET LOWER ACCUM1
         MUL              MULTIPLY A REG TIMES B REG
         ADDD    ACCUM3+1,PCR ADD ACCUM3 TO RESULT
         STD     ACCUM3+1,PCR STORE RESULT IN ACCUM3
         BCC     MUL322     NO CARRY
         INC     ACCUM3,PCR ADD CARRY TO ACCUM3

MUL322   LDA     ACCUM2,PCR GET UPPER ACCUM2
         LDB     ACCUM1,PCR GET UPPER ACCUM1
         MUL              MULTIPLY A REG TIMES B REG
         ADDD    ACCUM3,PCR ADD ACCUM3 TO RESULT
         STD     ACCUM3,PCR STORE RESULT IN ACCUM3
         RTS              RETURN


*        DIVIDE ACCUM3 BY 252 SUBROUTINE

*        ACCUM3 = REMAINDER
*        ACCUM4 = QUOTIENT


DIV252   LDA     #252       GET DIVISOR
         STA     -8,S       STORE IN STACK
         CLR     -7,S       CLEAR STACK
         CLR     -6,S       CLEAR STACK
         CLR     -5,S       CLEAR STACK
         LDA     #1         GET MASK BIT
         STA     -4,S       STORE IN STACK
         CLR     -3,S       CLEAR STACK
         CLR     -2,S       CLEAR STACK
         CLR     -1,S       CLEAR STACK
         CLR     ACCUM4,PCR CLEAR ACCUM 4
         CLR     ACCUM4+1,PCR CLEAR ACCUM 4
         CLR     ACCUM4+2,PCR CLEAR ACCUM 4
         CLR     ACCUM4+3,PCR CLEAR ACCUM 4

DIV1     LDA     -8,S       GET DIVISOR VALUE
         CMPA    ACCUM3,PCR COMPARE TO DIVIDEND
         BLO     DIV4       SUB DIVISOR FROM DIVIDEND
         BHI     DIV2       ROTATE DIVISOR AND MASK
         LDA     -7,S       GET DIVISOR VALUE
         CMPA    ACCUM3+1,PCR COMPARE TO DIVIDEND
         BLO     DIV4       SUB DIVISOR FROM DIVIDEND
         BHI     DIV2       ROTATE DIVISOR AND MASK
         LDA     -6,S       GET DIVISOR VALUE
         CMPA    ACCUM3+2,PCR COMPARE TO DIVIDEND
         BLO     DIV4       SUB DIVISOR FROM DIVIDEND
         BHI     DIV2       SHIFT RIGHT DIVISOR AND MASK
         LDA     -5,S       GET DIVISOR VALUE
         CMPA    ACCUM3+3,PCR COMPARE TO DIVIDEND
         BLS     DIV4       SUB DIVISOR FROM DIVIDEND

DIV2     LSR     -4,S       SHIFT RIGHT STACK
         ROR     -3,S       SHIFT RIGHT STACK
         ROR     -2,S       SHIFT RIGHT STACK
         ROR     -1,S       SHIFT RIGHT STACK
         BCC     DIV3       SHIFT RIGHT STACK
         RTS                RETURN

DIV3     LSR     -8,S       SHIFT RIGHT STACK
         ROR     -7,S       SHIFT RIGHT STACK
         ROR     -6,S       SHIFT RIGHT STACK
         ROR     -5,S       SHIFT RIGHT STACK
         BRA     DIV1       COMPARE DIVISOR TO DIVIDEND

DIV4     LDA     ACCUM3+3,PCR GET ACCUM 3
         SUBA    -5,S       SUBTRACT STACK
         STA     ACCUM3+3,PCR STORE ACCUM 3
         LDA     ACCUM3+2,PCR GET ACCUM 3
         SBCA    -6,S       SUBTRACT STACK
         STA     ACCUM3+2,PCR STORE ACCUM 3
         LDA     ACCUM3+1,PCR GET ACCUM 3
         SBCA    -7,S       SUBTRACT STACK
         STA     ACCUM3+1,PCR STORE ACCUM 3
         LDA     ACCUM3,PCR GET ACCUM 3
         SBCA    -8,S       SUBTRACT STACK
         STA     ACCUM3,PCR STORE ACCUM 3
         LDA     ACCUM4+3,PCR GET ACCUM 4
         ADDA    -1,S       ADD STACK
         STA     ACCUM4+3,PCR STORE ACCUM 4
         LDA     ACCUM4+2,PCR GET ACCUM 4
         ADCA    -2,S       ADD STACK
         STA     ACCUM4+2,PCR STORE ACCUM 4
         LDA     ACCUM4+1,PCR GET ACCUM 4
         ADCA    -3,S       ADD STACK
         STA     ACCUM4+1,PCR STORE ACCUM 4
         LDA     ACCUM4,PCR GET ACCUM 4
         ADCA    -4,S       ADD STACK
         STA     ACCUM4,PCR STORE ACCUM 4
         BRA     DIV2       SHIFT RIGHT DIVISOR & MASK

*****************************************************
*                                                   *
*              ISAM SUBROUTINE EXIT                 *
*                                                   *
*****************************************************


EXIT     RTS                RETURN

ERR100   LDD     #100       INVALID FILE NUMBER
         RTS                RETURN

ERR105   LDD     #105       INVALID COMMAND

         RTS                RETURN

ERR110   LDD     #110       INVALID FILE SPECS
         RTS                RETURN

ERR115   LDD     #115       INVALID FILE VERSION
         RTS                RETURN

ERR120   LDD     #120       INVALID RECORD SIZE
         RTS                RETURN

ERR125   LDD     #125       RECORD SIZE TO LARGE
         RTS                RETURN

ERR130   LDD     #130       INVALID FILE SIZE
         RTS                RETURN

ERR135   LDD     #135       FILE SIZE TO LARGE
         RTS                RETURN

ERR140   LDD     #140       ACTUAL FILE SIZE > MAX
         RTS                RETURN

ERR145   LDD     #145       MAX SECTORS EXCEEDED
         RTS                RETURN

ERR150   LDD     #150       CORRUPT FILE
         RTS                RETURN

ERR155   LDD     #155       FILE NOT OPEN
         RTS                RETURN

ERR160   LDD     #160       FILE IS OPEN
         RTS                RETURN

ERR165   LDD     #165       START OF FILE
         RTS                RETURN

ERR170   LDD     #170       END OF FILE
         RTS                RETURN

ERR175   LDD     #175       CANT ADD NULL RECORD
         RTS                RETURN

ERR180   LDD     #180       FILE FULL
         RTS                RETURN

FMSERR   CLRA               CLEAR A REGISTER
         LDB     1,X        GET FCB STATUS BYTE
         RTS                RETURN

*****************************************************
*                                                   *
```

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Media.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

K-BASIC now makes the multitude of TSC XBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and WIN!! -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (OS-9 or FLEX), Including the Assembler

!!! Special !!! ~~$199.00~~ $ 99.49

SAVE $100.00

SCULPTOR

!!! New Prices !!!

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a **FAST** Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. PLUS, the Application can also be run on MS-DOS and Unix machines! Sculptor handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes ISAM File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time License. Additional charge for Networked Units. Prices for Development Package/Run-time. Discounts available for purchases of 5 or more Run-time Packages.

OS-9 / UniFLEX --			68000 UniFLEX --			MS DOS --	
IBM PC Zenix --	$995.00/$175.00		Altos Zenix --	$1595.00/$265.00		--	$595.00/$115.00
MS DOS Network --	*	**	UNIX --	*	**	PC DOS --	* **

* Full Developement Package ** Run Time Package Only Full OEM and Dealer Discounts Available!

!!! Special Buy Out !!! **SPECIAL** - Limited Quantity !!! Special Buy Out !!!

6809 FLEX SOFTWARE		
TSC Flex Utilities	was $75.00	Buy only $50.00
TSC Basic	was $75.00	Buy only $35.00
TSC Extended Precompiler	was $50.00	Buy only $35.00
TSC Text Processor	was $75.00	Buy only $50.00
TSC Flex Precompiler	was $50.00	Buy only $35.00
TSC Flex Basic	was $75.00	Buy only $50.00
TSC Flex Diagnostic	was $75.00	Buy only $50.00
TSC Text Processor	was $75.00	Buy only $50.00
TSC Assembler	was $50.00	Buy only $35.00
TSC Debug	was $75.00	Buy only $50.00
TSC Precompiler	was $50.00	Buy only $35.00
TSC Editor	was $50.00	Buy only $35.00
TSC Sort/Merge	was $75.00	Buy only $50.00
TSC Utilities	was $75.00	Buy only $50.00
DISKS ONLY		
TSC Extended Precompiler	was $50.00	Buy only $35.00
TSC Basic Flex	was $75.00	Buy only $50.00
TSC Diagnostics	was $75.00	Buy only $50.00
TSC Utilities	was $75.00	Buy only $50.00

MANUALS ONLY		
TSC Basic Precompiler	was $25.00	Buy only $20.00
TSC Text Editor	was $25.00	Buy only $20.00
TSC Debug	was $25.00	Buy only $20.00
TSC Mnemonic Assembler	was $25.00	Buy only $20.00
COMPLETE 6800 SOFTWARE		
TSC Text Processor	was $50.00	Buy only $35.00
TSC Precompiler	was $50.00	Buy only $35.00
TSC Diagnostics	was $75.00	Buy only $50.00
DISKS ONLY - 6800 Software		
TSC Editor	was $50.00	Buy only $35.00
TSC Utilities	was $100.00	Buy only $70.00
TSC Assemblers	was $50.00	Buy only $35.00
MANUALS ONLY - 6800 Software		
TSC Diagnostics	was $25.00	Buy only $20.00
TSC Debug	was $25.00	Buy only $20.00
TSC Text Processor	was $25.00	Buy only $20.00

!!! Please Specify Your Operating System & Disk Size !!!

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the **TSC Macro Assembler.** F, CCF - **$99.95**

Macro Assembler for TSC -- The FLEX **STANDARD** Assembler.
Special -- CCF **$35.00**; F **$50.00**

OSM Extended 6809 Macro Assembler from **Lloyd I/O.** -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 **$99.00**

Relocating Assembler w/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers. F,CCF **$150.00**

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - **$75.00**

XMACE — MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - **$98.00**

TRUE CROSS ASSEMBLERS from **Computer Systems Consultants** -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - **$50.00**
any 3 - **$100.00**
the complete set **w/ C Source** (except the 68000 Source) - **$200.00**

XASM Cross Assemblers for **FLEX** from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - **$150.00**

CRASM8 from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- **$399.00**

CRASM 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000.**
FLEX, CCF, OS-9 **$249.00**

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-5D Bas (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only **$49.00**	F, **$99.00**
CCF, Obj. Only **$50.00**	U, **$100.00**
CCF, w/Source **$99.00**	O, **$101.00**
CCD, Obj. Only **$50.00.**	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only **$100.00**	CCD, Obj. Only **$ 59.95**
F, " " **$100.00**	O, " " **$150.00**
	U, " " **$300.00**

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including Interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. **500+** page Manual with tutorial guide. F, CCF - **$198.00**

MMINSICAL from **Whimsical Developments** -- Now supports **Real Numbers.** "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, **6-digit Real Numbers,** unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - **$195.00**

C Compiler from **Windrush Micro Systems** by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - **$295.00**

C Compiler from **Intrel** -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
F, CCF, O, and U - **$575.00**

PASCAL Compiler from **Lucidata** -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - **$190.00** F 8" - **$205.00**

PASCAL Compiler from **Compukit** (now **Certified Software**) -- For the **PROFESSIONAL;** ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.
F and CCF - **$425.00** One Year Maint. - **$100.00**

K-BASIC from **LLOYD I/O** -- A "Native Code" **BASIC Compiler** which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, **streamlined,** Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler **with** Assembler - **$199.00**

CRUNCH COBOL from **Compusense Ltd.** -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- **$99.95**

FORTH from **Stearns Electronics** -- A CoCo FORTH Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to **disk.** Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer **ONLY** - **$58.95**

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

*FLEX is a trademark of Technical Systems Consultants
™OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

SOFTWARE DEVELOPMENT

BasicO9 XRef from **Southeast Media** -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each lin numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.

O & CCO obj. only -- **$39.95**; w/ Source - **$79.95**

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File: view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

P, CCF -- **RACE Utility** 5" - **$40.00**, 8" - **$50.00**

DUB from **Southeast Media** -- A **UniFLEX** "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled** basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - **$219.95**

FULL SCREEN FORMS DISPLAY from **Computer Systems Consultants** -- **TSC Extended BASIC** program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F and CCF, U - **$25.00**, w/ Source - **$50.00**

DISK UTILITIES

OS-9 VDisk from **Southeast Media** -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

-- **Level I ONLY** -- OS-9 obj. only - **$79.95**; w/ Source - **$149.95**

O-F from **Southeast Media** -- Written in **BASICO9 (with Source)**, includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an **OS-9** disk to **FLEX** format so it can be used normally by FLEX; and FLEX, a BASICO9 Program that does the actual read or **write** function to the special **O-F Transfer Disk**; user-friendly menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. **SPECIAL 60 DAY OFFER** O-**$39.95**

COPYMULT from **Southeast Media** -- Copy LARGE Disks to several **smaller** disks. FLEX utilities allow the backup of **ANY** size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included.

ALL 4 Programs (FLEX, 8" or 5") **$99.50**

COPYCAT from **Lucidata** -- Pascal **NOT** required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in **Modular Source Code** (Assembly Language) to help solve unusual problems.

F and CCF 5" - **$50.00** F 8" - **$65.00**

FLEX DISK UTILITIES from **Computer Systems Consultants** -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors**; **Test Disk** for errors; **Compare two Disks**; a fast **Disk Backup Program**; **Edit Disk Sectors**; **Linearize Free-Chain** on the Disk; print **Disk Identification**; and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Ten XBASIC Programs including: A **BASIC Resequencer** with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare**, **Merge**, or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers**, compare **two resequenced files**, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC **Cross-Reference** Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in **TSC BASIC**, **EBASIC**, and **PUNCOMPILED BASIC Programs.**

ALL Utilities **include Source** (either BASIC or A.L. Source Code).

F and CCF - **$50.00**

BASIC Utilities ONLY for UniFLEX -- **$30.00**

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

FLEX, CCF, OS-9, UniFLEX; with complete Source - **$100.00**
without Source - **$50.00**

XDATA from **Southeast Media** -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - **$299.99**

GAME

RAPIER - 68Dg Chess Program from **Southeast Media** -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the **strongest** CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - **$79.95**

VISA MasterCard

**** SHIPPING ****
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware

!!! Please Specify Your Operating System & Disk Size !!!

BUSINESS

WORD PROCESSING

SCREDITOR III from **Windrush Micro Systems** -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from **Great Plains Computer Co.** -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES --> CCF and CCO - $99.95, F or O - $179.95, U - $249.95

STYLO-SPELL from **Great Plains Computer Co.** -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES --> CCF and CCO - $69.95, F or O - $99.95, U - $249.95

STYLO-MERGE from **Great Plains Computer Co.** -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES --> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from **Southeast Media** -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the **Display**. Use the **PPRINT.CMD** supplied for producing multiple copies of the "Formatted" Text on the Printer **INCLUDING IMBEDDED PRINTER CONTROLS** (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson HX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:

Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: **FLEX** - CC
Disk #2: JUSTSC object and source in C: **FLEX** - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax **JUST.CMD** object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent **BUDGET PRICED** word processor, **with features none of the others have.** This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - **$49.95**
Disk Set (2) - F & CCF & OS9 (C version) - $69.95

SPELL8 "Computer Dictionary" from **Southeast Media** -- **OVER 120,000 words!** Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELL8 first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELL8 also allows the use of **Small Disk Storage systems.**

F and CCF - $129.95

DATA BASE - ACCOUNTING

XDMS from **Westchester Applied Business Systems** -- Powerful DBMS; M.I. program will work on a **single sided 5" disk**, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for **Giant** Data Bases. **XDMS Level I** provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. **XDMS Level II** adds the POWERFUL "**GENERATE**" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. **XDMS Level III** adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System **Manual** - $24.95
XDMS Lvl I - F & CCF - **$129.95**
XDMS Lvl II - F & CCF - **$199.95**
XDMS Lvl III - F & CCF - **$269.95**

ACCOUNTING PACKAGES -- **Great Plains Computer Co.** and **Universal Data Research, Inc.** both have Data Base and Business Packages written in TSC XBASIC for FLEX, CoCo FLEX, and UniFLEX.

MISCELLANEOUS

TABULA RASA SPREADSHEET from **Computer Systems Consultants** -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires **TSC's Extended BASIC.**

F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC from **Computer Systems Center** -- Electronic Spread Sheet for the 6809.

F and SPECIAL CCF - $200.00. U - $395.00

FULL SCREEN INVENTORY/MRP from **Computer Systems Consultants** -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires **TSC's Extended BASIC.**

F and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from **Computer Systems Consultants** -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or labels, etc. Requires **TSC's Extended BASIC.**

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from **Southeast Media** -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $59.95, U - $89.95

Availability Legend --
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

```
*****************************************************
*                                                   *
*                  DATA STORAGE                     *
*                                                   *
*****************************************************

IFVER    FCC       'ISAM00001'          ISAM FILE VERSION


*****************************************************
*                                                   *
*                INSTALL PARAMETERS                 *
*                                                   *
*****************************************************

INPAR1   RMB       2          MAX NUMBER OF FILES
INPAR2   RMB       2          MAX RECORD SIZE
INPAR3   RMB       2          MAX FILE SIZE
```

```
*****************************************************
*                                                   *
*                  VARIABLE STORAGE                 *
*                                                   *
*****************************************************

COUNT    RMB       2          GENERAL PURPOSE COUNTER
RECONT   RMB       2          REORG RECORD COUNTER
CIFANT   RMB       2          CALLER ISAM FCB POINTER
LLIMIT   RMB       2          LOWER RECORD LIMIT
ULIMIT   RMB       2          UPPER RECORD LIMIT
ACCUM1   RMB       2          ACCUMULATOR NUMBER 1
ACCUM2   RMB       2          ACCUMULATOR NUMBER 2
ACCUM3   RMB       4          ACCUMULATOR NUMBER 3
ACCUM4   RMB       4          ACCUMULATOR NUMBER 4
BUFFER

*****************************************************

         END       INSTAL
```

Bit Bucket

U N I M A C S

Travel Management Control Systems

Dear Don,

Having been involved with micro's since 76, (I'm an old timer I guess), I have wanted to have a good reason to tell you how much your magazine has meant to me and how it has helped me get where I am today, but that's a long story. Instead I'll just say thank you very much, keep up the excellent work.

Enclosed are two upgrades that I made to the FUNCTION programs that David Goadby sent to you as shown on page 46 of the FEB issue of 68 MICRO. I have up-dated the two programs to make the FUNCTION program relocatable. It first relocates itself under MEMEND, then moves memend to protect itself and then link into the input routines as the original program did. The FUNLOAD program finds the function key table and loads a new table into it from a text file.

These are nothing dramatic but there are a couple of things worth noting: first, the function keys will return whatever they contain, to the calling program. In case people missed it, this means that a date can be put into a function and returned to a Basic or any other applications program. This is in addition to returning a command to FLEX/STAR-DOS as the writer originally intended. The second thing that is notable is that the programs can save lots of work if you are a professional programmer as I am. I must recommend strongly to everyone that the hour or so to enter and assemble the two programs is well worth the while.

If you examine the FUNCTION program, you will see that the functions that I made resident are for XDMS, which is the greatest Database Management system ever written. I am using it in an extensive commercial application and would be glad to write a review of it if you are interested. Also included is a sample function set file called FUNSET1.TXT. I set this version up so that I have all the tools available for assembly language programming.

I did not take the time to optomize the above programs with the X's and U registers, but it is an easy exercise for those interested and would shorten them a great deal.

Best regards!!!

Jim Cerwitz
7907 E Wood Dr
Scottsdale AZ 85260
602-948-9304

```
                        OPT     PAG
                *****************************************
                * RELOCATABLE CODE VERSION OF
                * PROGRAM: FUNCTION
                * FEB 1985 68MICRO PG 46
                * AUTHOR: DAVID GOADBY
                * REVISED BY: JIM CERWITZ
                *****************************************

                * SPECIFIC EQUATES

                ****   SYMBOLS ****

           0028 LINEL   EQU     40          maximum command line length
           0009 TABCHR  EQU     $9          TAB CHARA
           000D CR      EQU     $0D         CARR RET
           000A LF      EQU     $0A         LINE FEED
           0003 VN      EQU     $3          version 3

                **** LOCATIONS WITHIN FLEX ****

           CC00 BACK    EQU     $CC00       backspace chara
           CD03 WARMB   EQU     $CD03       warm start address
           CD09 FINDI   EQU     $CD09       INCH vector
           CD0C FINDO   EQU     $CD0C       INCH2 vector
           CD0F OUTPUT  EQU     $CD0F       OUTCH vector
           CD12 OUTPO   EQU     $CD12       OUTCH2 vector
           CD24 CRLF    EQU     $CD24       FLEX PCRLF
           CC2B FLMEM   EQU     $CC2B       FLEX memtop

                *** program to relocate operating portion ***

C100                    ORG     $C100       FLEX UTILITY AREA

C100 20 05      START   BRA     RUN         branch around temps
C102 03                 FCB     VN          version no

C103            TEMPD   RMB     2
C105            NUSTAT  RMB     2

C107 BE CD0A    RUN     LDX     FINDI+1     get orig FLEX INCH address
C10A AF 8D 0058         STX     INCHFL,PCR  save it
C10E BE CD10            LDX     OUTPI+1     get orig OUTCH vector
C111 AF 8D 005J         STX     OUTFL,PCR   save orig FLEX OUTCH vector
C115 BE CC2B            LDX     FLMEM       get memory end into X
C118 AF 8D 0048         STX     MEMTOP,PCR  save orig memtop

C11C 10BE C38A  MOVE    LDY     #ZZEND+1    get end of pgm +1
C120 8E C149            LDX     #START      get start of pgm
C123 AF 8C 0D           STX     TEMPD,PCR   save start for ref
C126 BE 0C2B            LDX     FLMEM       get top of user memory
C129 30 01              LEAX    1,X         inc by 1 for pre-decrement
C12B A6 A2      RELOC   LDA     0,-Y        get 1st byte
C12D A7 82              STA     0,-X        save in new location
C12F 10AC 8C D0         CMPY    TEMPD,PCR   at end of pgm?
C133 26 F6              BNE     RELOC       no, loop for next byte
C135 AF 8C C0           STX     NUSTAT,PCR save new pgm start

C138 30 1F              LEAX    -1,X        new top of user memory
C13A BF 0C2B            STX     FLMEM       set it & protect pgm

                *** CODE RELOCATED NOW HOOK INCH ***

C13D AE 8C C5           LDX     NUSTAT,PCR get new INCH
C140 BF CD0A            STX     FINDI+1     hook it in
C143 BF CD00            STX     FINDO+1     here too

                *** HOOKS DONE, CODE MOVED ***

C146 7E CD03            JMP     WARMB

                *** MAIN PROGRAM HERE ***

C149 34         START   PSHS    B,X         save registers
C14B 6D 8D 0014         TST     FLAG,PCR    diversion still on?
C14F 26 29              BNE     FINPT       yes
C151 AD 9D 0011 START1  JSR     [INCHFL,PCR] no, act normal
C155 81 09              CMPA    #TAB        want function key?
C157 27 11              BEQ     TRAP1       yes
C159 AF 8D 0003 RETURN  STX     #SAVE1,PCR
C15D 35 14              PULS    B,X         recover registers
C15F 39                 RTS
```

```
                        *** temp storage here ***
C160                    XSAVE1  RMB     2       function location pointer
C162                    FNUM    RMB     1       requested function number
C163 00                 FLAG    FCB     0       diversion flag
C164                    HEATOP  RMB     2       original FLEX HEATOP value
C166                    INCHFL  RMB     2       orig FLEX INCH= call
C168                    OUTFL   RMB     2       ORIG FLEX OUTCH VECTOR

C16A AD  9C F9          TRYPL   JSR     [INCHFL,PCR] get next char
C16D 17  00A0                   LBSR    BSPACE  remove it
C170 81  30             EDITCK  CMPA    #'0     want to edit?
C172 27  25                     BEQ     EDIT    go edit
C174 8D  79                     BSR     VALID   get address of string
C176 25  D9                     BCS     START1
C178 63  8C E8                  COM     FLAG,PCR  set diversion flag
C17B AE  8C E2          FINPT   LDX     XSAVE1,PCR save location of next char
C17E A6  84                     LDA     0,X     get next char
C180 81  00                     CMPA    #CR     is it the end?
C182 26  05                     BNE     CHKNXT  if not, continue
C184 6F  8C DC          FUNCRT  CLR     FLAG,PCR  clear diversion flag
C187 20  D0                     BRA     RETURN
C189 81  5C             CHKNXT  CMPA    #'\     input from operator needed?
C18B 26  05                     BNE     DISPL1  no
C18D 6F  8C D3                  CLR     FLAG,PCR  set to normal
C190 20  BF                     BRA     START1  get user input
C192 AD  9C D3          DISPL1  JSR     [OUTFL,PCR] send char to terminal
C195 30  01                     LEAX    1,X     point to next entry
C197 20  C0                     BRA     RETURN

C199 AD  9C CA          EDIT    JSR     [INCHFL,PCR] get function no
C19C 80  72                     BSR     BSPACE
C19E 8D  4F                     BSR     VALID   see if 1-9 & get address
C1A0 25  AF                     BCS     START1  error
C1A2 AF  8C BB                  STX     XSAVE1,PCR save entry value

                        * DISPLAY FUNCTION KEYS

C1A5 86  66             DISPLAY LDA     #'F
C1A7 AD  9C BE                  JSR     [OUTFL,PCR] send to terminal
C1AA A6  8C B5                  LDA     FNUM,PCR  get function number
C1AD AD  9C B0                  JSR     [OUTFL,PCR] send to term
C1B0 86  3D                     LDA     #'=
C1B2 AD  9C A3                  JSR     [OUTFL,PCR] send to term
C1B5 AE  8C AB                  LDX     XSAVE1,PCR point to location
C1B8 A6  80             PLOOP   LDA     0,X+    get char
C1BA 81  0D                     CMPA    #CR     is it last char in function

C1BC 27  09                     BEQ     EDNEXT  go get new function
C1BE AD  9C A7                  JSR     [OUTFL,PCR] no so print it
C1C1 20  F5                     BRA     PLOOP   loop for more
C1C3 86  0D             START2  LDA     #CR     force CR
C1C5 20  92                     BRA     RETURN

                        * NOW CK FOR R

C1C7 86  3A             EDNEXT  LDA     #':     seperator
C1C9 AD  9C 9C                  JSR     [OUTFL,PCR] send to term
C1CC AD  9C 97                  JSR     [INCHFL,PCR] get option
C1CF 81  52                     CMPA    #'R     want to replace?
C1D1 26  F0                     BNE     START2  no go exit

                        * ENTER STRING

C1D3 86  3D                     LDA     #'=     prompt
C1D5 AD  9C 90                  JSR     [OUTFL,PCR] prin it
C1D8 AE  8C 85                  LDX     XSAVE1,PCR point to where it goes
C1DB C6  27                     LDB     #LINEL-1  get line length-1
C1DD AD  9C A6          EDLOOP  JSR     [INCHFL,PCR] get chars from operator
C1E0 A7  80                     STA     0,X+    save it in table
C1E2 81  0D                     CMPA    #CR     was it a CR?
C1E4 27  07                     BEQ     DONE    yes
C1E6 5A                         DECB            no reduce available space
C1E7 26  F4                     BNE     EDLOOP  end of function?
C1E9 86  0D                     LDA     #CR     insert CR
C1EB A7  80                     STA     0,X+    go it
C1ED 20  D4             DONE    BRA     START2  exit

                        *
                        * THIS ROUTINE VALIDATES THE KEY
                        * NUMBER AND RETURNS WITH X POINTING
                        * TO THE STRING ADDRESS.
                        *

C1EF A7  8D FF6F        VALID   STA     FNUM,PCR  save function no
C1F3 81  30                     CMPA    #'0     less than 0?
C1F5 23  16                     BLS     ERROR   0 not allowed
C1F7 81  39                     CMPA    #'9     greater than 9?
C1F9 22  12                     BHI     ERROR   >9 not allowed
C1FB 80  31                     SUBA    #'1     SET TO BINARY-1 adjust
C1FD 30  8D 0018                LEAX    TABLE,PCR point to location in table

                        * MULTIPLY BY LINEL TO INDEX INTO TABLE
                        * THEN ADD TO X TO POINT TO ENTRY

C201 C6  28                     LDB     #LINEL  get line length
C203 3D                         MUL             A*B->D
C204 30  8B                     LEAX    D,X     X+D->D
C206 AF  8D FF56                STX     XSAVE1,PCR save location
C20A 1C  FE                     CLC             clear carry
C20C 39                         RTS             return
C20D 1A  01             ERROR   SEC             set carry if error
C20F 39                         RTS

C210 34  02             BSPACE  PSHS    A       save chars in acc A
C212 86  CC00                   LDA     BACK    get FLEX backspace chars
C215 AD  9D FF4F                JSR     [OUTFL,PCR] send to term
C219 35  02                     PULS    A       restore chars
C21B 39                         RTS             return

                        * TABLE OF PRESET COMMANDS

C21C 58 44 4D 53        TABLE   FCC     /XDMS/  command 1
C220 00                         FCB     CR

C244                            ORG     14*LINEL+TABLE
C244 44 45 46 49                FCC     /DEFINE \/ command 2
C248 4E 45 20 5C
C24C 00                         FCB     CR

C26C                            ORG     29*LINEL+TABLE
C26C 55 50 44 41                FCC     /UPDATE \/ command 3
C270 54 45 20 5C
C274 00                         FCB     CR

C294                            ORG     39*LINEL+TABLE
C294 47 45 4E 45                FCC     / GENERATE \/ command 4
C298 52 41 54 45
C29C 20 5C
C29E 00                         FCB     CR

C29C                            ORG     44*LINEL+TABLE
C2BC 2F 20 47 45                FCC     /# GENERATE,\/ command 5
C2C0 4E 45 52 41
C2C4 54 45 2C 5C
C2C8 00                         FCB     CR

C2E4                            ORG     50*LINEL+TABLE
C2E4 45 44 49 54                FCC     /EDIT \ / command 6

C2E8 20 5C 20
C2EB 00                         FCB     CR

C30C                            ORG     66*LINEL+TABLE
C30C 44 49 52 20                FCC     /DIR .FMU .CIL .DMS/ command 7
C310 2E 46 4D 55
C314 20 2E 43 49
C318 4C 20 2E 44
C31C 4D 53
C31E 0D                         FCB     CR

C334                            ORG     76*LINEL+TABLE
C334 4C 49 53 54                FCC     /LIST \/  command 8
C338 20 5C
C33A 00                         FCB     CR

C35C                            ORG     84*LINEL+TABLE
    51 55 49 54                 FCC     /QUIT/    command 9
C360 00                         FCB     CR

C384                            ORG     94*LINEL+TABLE
C384 00                         FCB     00

                C389    ZZEND   EQU     *

                                END     START2

0 ERROR(S) DETECTED
```

SYMBOL TABLE:

BACK	CC00	BSPACE	C210	CHKNXT	C189	CR	0000	CRLF	CD46
DISPL1	C192	DISPLA	C1A5	DONE	C1ED	EDIT	C199	EDITCK	C170
EDLOOP	C1DD	EDNEXT	C1C7	ERROR	C20D	FINP1	C009	FINP2	C00C
FINPT	C17B	FLAG	C163	FUNCRT	CC23	FNUM	C162	FUNCRT	C184
INCHFL	C166	LF	000A	LINEL	0028	MAXTOP	C164	MOVE	C11C
NXBYTE	C105	OUTFL	C168	OUTP1	CD0F	OUTPUT	CD12	PAUSE	C108
RELOC	C1E9	RETURN	C159	RUN	C107	START1	C149	START2	C1B1
START2	C1C3	STARTZ	C100	TABLE	C21C	TRYPR	C103	TRAP1	C16A
TWIDTH	0009	VALID	C1EF	VN	0003	WARMS	CD03	XSAVE1	C160
ZZEND	C383								

```
                        OPT     PAG
                ********************************************
                * PROGRAM: FUNLOAD.TXT
                * AUTHOR:  DAVID GOADBY
                * CREATED: 3-24-85
                * REVISED BY: JIM GERWITZ
                ********************************************

                        * SPECIAL EQUATES

                0028    LINEL   EQU     40      max line length

                        * FLEX EQUATES

                C840    FCB     EQU     $C840
                D406    FMS     EQU     $D406
                CD03    WARMS   EQU     $CD03
                CD2D    GETFIL  EQU     $CD2D
                CD3F    RPTERR  EQU     $CD3F
                0003    FMSCLS  EQU     $0003
                CD33    SETEXT  EQU     $CD33
                CD1E    PSTRNG  EQU     $CD1E
                CD09    FINP1   EQU     $CD09
                0002    VN      EQU     2       VERSION
                0003    OFFSET  EQU     $D3     OFFSET TO TABLE

                        * PROGRAM BEGINS HERE

                                ORG     $C100

C100                    START   BRA     START1
C102 02                         FCB     VN
C103 8E  C840           START1  LDX     #FCB
C106 BD  CD2D                   JSR     GETFIL
C109 25  78                     BCS     ERROR
C10B 86  01                     LDA     #$1
C10D A7  84                     STA     0,X
C10F BD  CD33                   JSR     SETEXT
C112 BD  D406                   JSR     FMS
C115 26  6C                     BNE     ERROR

                        * SETUP TABLE PARAMETERS

C117 9E  C009                   LDX     FINP1+1
C119 C6  03                     LDB     #OFFSET
C11C 3A                         ABX
C11D AF  8D 0084                STX     TABSTA,PCR SAVE TABLE START
C121 AF  8D 0083                STX     CURPOS,PCR
C125 108E 0168                  LDY     #14*LINEL
C129 10AF 8D 0088               STY     EOFTAB,PCR
C12E C6  28                     LDB     #LINEL
C130 3A                         ABX
C131 AF  8D 009F                STX     EOFLIN,PCR
C135 C6  09                     LDB     #9
C137 E7  8D 009F                STB     LINECT,PCR

                        * READ TABLE FROM DISK FILE

C13B 8E  C840           READLP  LDX     #FCB
C13E BD  D406                   JSR     FMS
C141 26  49                     BNE     ERROR1
C143 9E  8D 0091        STORLP  LDX     CURPOS,PCR
C147 AC  8D 0089                CPX     EOFLIN,PCR
C14B 27  0C                     BEQ     LINEND
C14D A7  80                     STA     0,X+
C14F AF  8D 0085                STX     CURPOS,PCR
C153 81  0D                     CMPA    #$0D
C155 26  E4                     BNE     READLP
C157 20  EA                     BRA     STORLP

                        * END OF LINE PROCESSING

C159 A6  1F             LINEND  LDA     -1,X
C15B 81  0D                     CMPA    #$0D
C15D 26  36                     BNE     ERROR2
C15F 8E  8D 0071                LDX     EOFLIN,PCR
```

```
78   C163 AF   8D 0071          BTX    CURPOS,PCR
79   C167 C6   28                LDB    #LINEL
80   C169 3A                     ABX
81   C16A AF   8D 0066          BTX    EOFLIN,PCR
82   C16E 6A   8D 0068          DEC    LINECT,PCR
83   C172 26   C7               BNE    READLP
84   C174 86   04       ENDIT   LDA    #$4
85   C176 BE   C840             LDX    #FCB
86   C179 A7   84               STA    0,X
87   C17B BD   0406             JSR    FMS
88   C17E 26   03               BNE    ERROR
89   C180 7E   CD03             JMP    WARMS
90
91                      * ERROR PROCESSING
92
93   C183 BD   CD3F     ERROR   JSR    APTERR
94   C186 BD   D403             JSR    FMSCLS
95   C189 7E   CD03             JMP    WARMS
96
97                      * THE FOLLOWING ARE SOFT ERRORS
98
99   C18C AE   8D 0008  ERROR1  LDX    MSG1,PCR
100  C190 BD   CD1E     ERRER1  JSR    PSTRNG
101  C193 20   0F               BRA    ENDIT
102  C195 AE   8D 001E  ERROR2  LDX    MSG2,PCR
103  C199 20   F3               BRA    ERRExit
104
105                     * ERROR STRINGS
106
107  C19B 4C 65 53 53 53 MSG1   FCC    /LESS THAN 9 ENTRIES FOUND/
     C19F 20 54 48 41
     C1A3 4E 20 39 20
     C1A7 45 4E 54 52
     C1AB 49 45 53 20
     C1AF 46 4F 55 4E
     C1B3 44
108  C1B4 0D 0A                 FCB    $D,$A
109  C1B6 04                    FCB    $4
110  C1B7 4C 49 4E 45   MSG2   FCC    /LINE EXCEEDS 40 CHARACTERS/
     C1BB 20 45 58 43
     C1BF 45 45 44 53
     C1C3 20 34 30 20
     C1C7 43 48 41 52
     C1CB 41 43 54 45
     C1CF 52 53
111  C1D1 0D 0A                 FCB    $D,$A
112  C1D3 04                    FCB    $4
113
114                     * DATA STORAGE
115
116  C1D4               EOFLIN  RMB    2
117  C1D6               EOFTAB  RMB    2
118  C1D8               CURPOS  RMB    2
119  C1DA               LINECT  RMB    1
120  C1DB               TABSTA  RMB    2
121
122                        END    START

0 ERROR(S) DETECTED
```

```
SYMBOL TABLE:

CURPOS C1D8   ENDIT  C174   EOFLIN C1D4   EOFTAB C1D6   ERREXT C190
ERRDR  C183   ERROR1 C18C   ERROR2 C195   FCB    C840   FINDI  CD09
FMS    0406   FMSCLS D403   GETFIL CD2D   LINECT C1DA   LINEL  0028
LINEND C1 9   MSG1   C19B   MSG2   C1B7   OFFSET 0003   PSTRNG CD1E
READLP C13B   APTERR CD3F   SETEST CD33   START  C100   STARTI C103
STORLP C143   TABSTA C1DB   WN     0002   WARMS  CD03
```

```
FUNSET1.TXT


        1.00=EDIT \
        2.00=LIST \
        3.00=DDEL \
        4.00=DS
        5.00=P LIST
        6.00=COPY \
        7.00=JIMERASMB \
        8.00=P JIMBASMB \
        9.00=S
```

Dear Don,

Thank you for your letter regarding my submission to 68 Micro. I also would like to thank you for extending my subscription.

Since the time I submitted the revised version of FUNCTION.CMD to you, I have added several upgrades that readers may want to have. I will itemize them and if you feel there might be any interest, I will send you the new version to publish. First I added a check to see if FUNCTION was already resident and if so, a message is printed to tell the operator. Second I added an automatic feature to read the date from the system Month Day Year registers and place this date as an ASCII value into function 09. Third, at the request of a friend that had a special need, I added the ability to imbed the "@" character in the function and have it perform the same as EOL, normally ";". Last, I modified the the way the edit function operates. To edit one of the functions, you enter TAB, then the function number. The function which is stored will then be displayed. If you wish to change it, you just type in the desired statement. If you change your mind, entering a <CR> will abort the edit. This was done because I could never remember to enter an "R" to signify replacement before editing.

The second reason I am writing results from your statement that you would appreciate any additional material that I have. It occured to me that some readers might be interested in the Payroll system that I am developing using XDMS. It would be difficult, if not impossible to just put it together as a big article and have it make sense. However the thought struck me, that if you were willing to give me a little guidance, I could turn it into a series of articles on how to implement a rather difficult subject using XDMS and AUTOTASK. I have no idea as to the ramifications of what I just said but at least it would be interesting and would be a chance to share some knowledge with other 68 Micro readers. I have a lot of software using these two programs such as a Check Register program to organize records by expense type and also balance a your checkbook.

Just for the record, I am using the PT-69 with my software in a vertical market situation and have been more than satisfied with Peripheral Technology's system. Also Fredrick Brown, the owner, has been of great assistance to me and the hardware has been the most reliable I have seen.

If you would be interested in persuing the thoughts above, please contact me either by letter or by phone.

Jim Gerwitz
7907 E. Wood DR
Scottsdale AZ 85260
Home: 602-948-9304

Expanding The MVME201 To 1 Megabyte

by Ray Robinson
Speech Research Centre
Macquarie University
North Ryde 2113
NSW Australia

INTRODUCTION

The MVME200 is a VME bus 64k byte memory card made by Motorola. The MVME201 is the same card, but with 256k byte capacity, due to 64Kx1 DRAMS (Dynamic RAMS) being used instead of 16Kx1 DRAMS. My VME bus system needed more memory, so I upgraded the card to 1M byte by using 256Kx1 DRAMS. Here's how I did it. I call it the MVME201-1.

ADDRESS

The memory is arranged physically as 2 banks of 18 chips each (36 total), giving a 16 bit word and 2 bits of parity.

These 2 banks are arranged as as 4 pages of memory 16 bits wide and individually addressable by 4 base address patch areas. The size of the page is set by the DRAMS used, 16K for 4116 (16Kx1), 64K for 4164 (64Kx1), and 256K for 41256 chips (256Kx1).

To modify the base address selection for 256K pages, add eight 10K resistors to the gates U45, U51, U55, and U59 on the Figure 5-2 sheet 4 of the MVME200/201 users manual. Also see Diagram 1 (below).

REFRESH

The 256Kx1 DRAMS require identicle refresh to the 64Kx1 DRAMS and so no modifications are required.

MULTIPLEXING

The MVME200 multiplexes 14 address lines to 7 for the 16Kx1 DRAMS. The MVME201 multiplexes 16 address lines to 8 for the 64Kx1 DRAMS. For the 256Kx1 DRAMS, we need 18 address lines multiplexed to 9. To add the mux for MA8 (the 9th address line to the 256Kx1 DRAMS), I used 3 spare gates and added 2 resistors.

The address lines LA16 and LA17 are gated with signal 100A and 100A* (an inverted 100A signal), by U80 and U81. U81 needs a 1K pullup resistor. The outputs from these 2 gates go to U69 and then through a 22 ohm matching resistor to the MA8 address line on the DRAMS (pin 1 or called VBB on 16Kx1 DRAMS). See Diagram 2 below. The signals 100A and 100A* are found on Figure 5-2 sheet 6 of the MVME200/201 users manual. The address lines LA16 and LA17 are on sheet 5. The destination for MA8 is on sheets 2, 8, and 9.

PROCEDURE

Remove 36 DRAMS. I added the 8 address pullup resistors to U45, U51, U55, and U59, by neatly soldering them on the component side of the board, from the appropiate pin to pin 16 (VCC) of the same chip. I did the same for the pullup resistor on U81 in the new mux. The spare gates on U80, U81, and U69 where checked with a multimeter to find out which pins were earthed. These were cut and bent under the chip as in Diagram 3 and checked again for any short circuits. Some black wirewrap wire was threaded neatly under chips and soldered directly to the pins of the gates, again on the component side of the board. One end of the 22 ohm matching resistor was soldered to the jumper W4 which is located near the front panel.

Add eighteen 256Kx1 DRAMS (150ns) to U1, U6, U11, U13, U18, U20, U24, U26, U31, U36, U38, U42, U46, U48, U52, U56, U60, and U62. They are $7 each in Australia.

PATCHING

Ensure the patches W5 and W2 are in, and W3, R6, W4, and W1 are open (Figure 5-2 sheet 2 MVME200/201 Users Manual). These configure the DRAMS and are located near the front panel. The 22 ohm resistor from the new mux (MA8) goes to the junction of R6 and W4.

The 4 memory pages are set with patches J2, J3, J4, and J5. Diagram 4 shows J2 patched for a base address of 000000 HEX and J3 for 040000 HEX. The other 2 pages (set by J4 and J5) are shown disabled. After testing, the extra 1/2 megebyte can be plugged in and enabled.

The other patches should remain unchanged. They are (as factory set) J1, J7, and J8 connected, and J6 open.

ERRORS

There are bypass capacitors on the old VBB line, now MA8 address line. They are C5, C14, C15, C25, C27, C38, C47, C54, and C61. Find these and remove them. Not all may have been fitted. Figure 5-2 sheet 2 of the MVME200/201 Users Manual wrongly shows all bypass caps on the +5 volt line.

Figure 5-2 sheet 5 of the manuals shows wrong pin labelling on one of the mux chips (U29). It should be U29 pin 3, LA08 and U29 pin 6, LA09 and U29 pin 10, LA10 and finally U29 pin 13, LA11.

TE

STING

The board was tested in my system, with the MVME101BUG monitor commands BI to initialise the RAM and BT to block test it from 10000 to 7FFFE. If the new mux (MA8 address line) is not working properly you will find you have only 64K of RAM with images throughout this range. If so, look for one of those bypass caps on the MA8 address line. The address space 0 to FFFF is used onboard by the MVME101 CPU and so the RAM on this card in that area is unused.

If all went well, you now have 1/2 a megabyte of RAM up to 80000 HEX. You can now boot VERSADOS and run SYSGEN and ASSEMBLER. I chose not to fit the other 1/2 megabyte of RAM until I need it. All you have to do is fit the chips and set the base address with J4 and J5 and the whole 1 megabyte is ready.

end

DIAGRAM 1

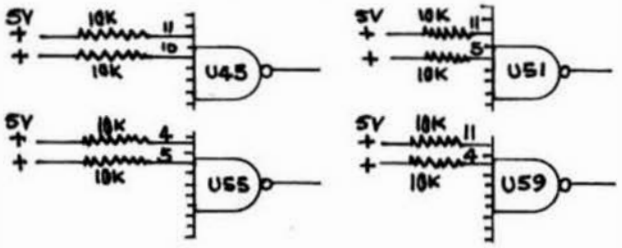


DIAGRAM 2

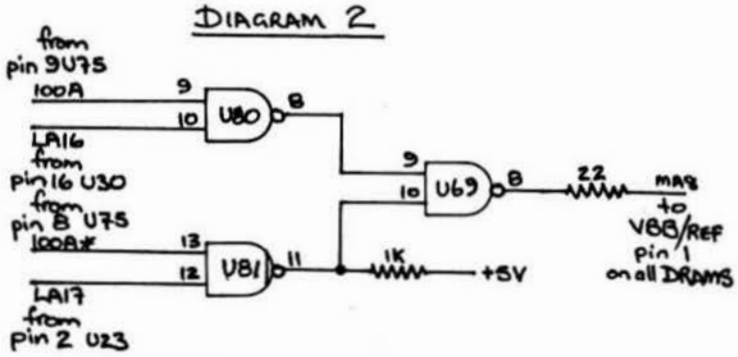


DIAGRAM 3

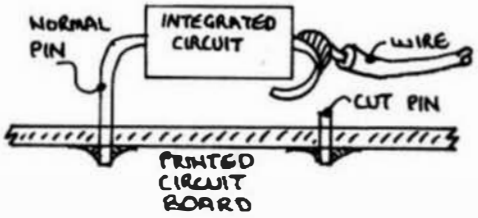
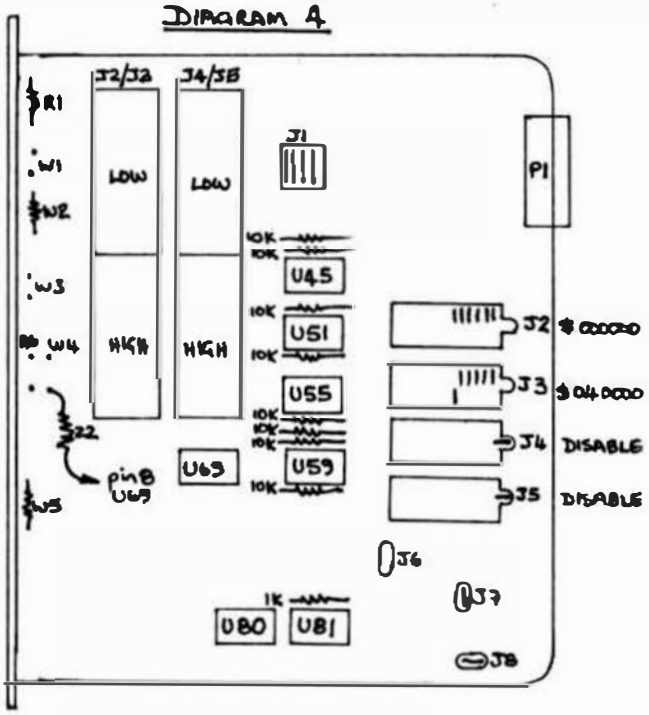


DIAGRAM 4



By: Troy Brumley
8552 Huddleston Drive
Cincinnati OH 45236

Using FLEX/Star-DOS

WHERE DID WE LEAVE OFF?

Try as I may, I just can't seem to get these articles out any faster than every other month at best. I am involved in several major projects at work so I'm pretty tired of computers when I get home. This has happened to me before, and I know it will pass, but in the meantime I won't be able to get these articles out monthly. Please bear with me.

As I recall, I promised to cover CoCo printer driver customization and I/O redirection in FLEX. I'll get to printer driver customization this month, but because of some other things I need to cover I'll have to ask you to wait for information on I/O redirection until the next article.

MY OBJECTIVES

If you've been reading Ron Anderson's FLEX USERS NOTES (you should be) you know that he is starting to review some basic FLEX concepts. We are working independently, and we are writing for separate audiences. From what he has been covering in the past few months I think that anyone who needs to learn how to program under FLEX should study his columns (I do).

The main group of people I am trying to help are new FLEX users who are somewhat familiar with RSDOS or another microcomputer system. I also hope to cover enough information to help prospective Advanced Operating System buyers choose between FLEX and OS/9. I don't have OS/9 yet, so I can't really compare them, but I can present FLEX for evaluation.

If you have any questions about FLEX, drop me a line at the address listed at the front of this article. If you want a personal answer, send a stamped and self addressed envelope. I can't promise quick turn around on letters that want personal responses, but I'll try to cover questions I get in these articles as soon as possible.

WHAT TO BUY WITH FLEX

I was talking with a friend of mine recently, and we were comparing notes on our computer systems. He has a CoCo, but he does not use FLEX, Star-DOS, or OS/9. He wondered why I bothered with FLEX, and what I could do with FLEX that he couldn't do with RSDOS. I showed him some of the things I do now, and some of the languages available under FLEX. My friend is a something of a language freak, so he became much more interested after I showed him some PL/9 listings in '68'.

Because FLEX is a mature operating system with a wide variety of languages and utilities available to you. An excellent word processor (STYLOGRAPH) and spread-sheet (DYNACALC) are reasonably priced. Almost all FLEX software will run on your CoCo. You can use your CoCo/FLEX system to learn a new programming language, develop utilities or applications software using those languages, do word processing, accounting, keep track of data, and so on.

I use my system for word processing, record keeping (I do some contract programming), and hobby related programming. I have K-BASIC (an excellent compiler), and I've started several projects in K-BASIC (all of which are on hold, unfortunately). Since I've read so many good things about PL/9 in '68' lately I'll probably save up enough to buy PL/9 for my next language.

As you can see, I am able to do everything I need with my little old CoCo (now that I've got FLEX).

FLEX and Star-DOS are both sold in simple packages. The operating system and the Utility Command Set (or UCS, I'll explain that soon) are about all you get. The CoCo versions of FLEX and Star-DOS include some additional utilities to allow you to copy files to and from your RSDOS disks.

If you plan on tinkering with FLEX, or doing any serious amount of programming, you will probably want an editor and assembler. FLEX can be purchased bundled with an editor and assembler. That's the FLEX CoCo SR. package that CPI sells. It's what I use, and I don't regret spending the extra money to get the editor and assembler.

You have to decide what you want to do with your computer before you buy FLEX. I can't think of too many things that I can't do with FLEX, but I know that there are a few things that I can't do with RSDOS. The way I see it, those of us with FLEX on our CoCos live in the best of worlds. Not only do we have access to extensive programming and applications software libraries (under both systems), we also have one of the nicest game playing computers around. No one with a GIMIX is flying Tom Mix's P-51 unless they also have a CoCo! Their GIMIX may be faster than our CoCo, but it doesn't have the graphics we do.

THE UTILITY COMMAND SET

This section is a summary of the FLEX Utility Command Set (UCS). The UCS is a standard set of programs to handle disk file management functions.

The standard commands in the UCS that TSC provides and their RSDOS equivalents are

FLEX	RSDOS
APPEND	-
ASN	DRIVE
BUILD	-
CAT	DIR
COPY	COPY
DATE	-
DELETE	KILL
EXEC	-
I	-
JUMP	EXEC
*LINK	-
LIST	-
NEWDISK	OSKINI
O	-
P	-
**PRINT	-
PROT	-
**QCHECK	-
RENAME	RENAME
SAVE	SAVEM
STARTUP	-
TTYSET	-
VERIFY	VERIFY
VERSION	-
XOUT	-

*LINK is called MAKESYS in the F-MATE FLEX conversion.
**PRINT and QCHECK are included with standard FLEX, but they don't work on most computers, so they aren't available with the F-MATE conversion.:

Obviously the UCS has several commands that have no real equivalent in RSDOS. Most of those commands that are unique to FLEX will be covered in later articles, since they relate to topics such as I/O redirection, terminal configuration, and debugging.

All of the commands in the UCS are DISK RESIDENT. This means that the must reside on the SYSTEM DRIVE (I'll explain that in the next article). Every time you use a command from the UCS it is loaded into a special section of memory in FLEX for execution. This allows the UCS to be expanded (there are several commands that come with F-MATE FLEX as part of the UCS that aren't available on other systems) and modified (recent issues of '68' have presented enhanced versions of the CAT command.

Because the commands must be loaded from disk every time they are used there is a slight delay in their execution. This is made up for by the substantial savings in memory. FLEX leaves 48K of memory available for user programs. RSDOS leaves you with less than 32K.

This dynamic loading of utilities is not unique to FLEX. OS-9 loads commands into memory unless they have already been loaded in. OS-9 leaves those commands in memory until you ask him to remove them. This can cause memory to get cluttered up rapidly. Both systems are a compromise for 8-bit computers, but both actually work very well. FLEX is certainly easier for me to use than MS-DOS on my PC at work.

There are a couple of 128K upgrade kits available that allow one. 64K bank of memory to be used as the standard 64K memory of the CoCo, and the other 128K bank to be used as a RAM DISK. This allows commonly used utilities and data files to be loaded into that 64K. When they are needed, they are loaded into memory from memory. This is done at incredible speed. Both FLEX and OS-9 level I benefit greatly from the use of a RAM DISK. If I ever get one, I'll let you know just how fast it really is.

USING A NON-STANDARD PRINTER

FLEX will let you define your printer one time, and it will remember how your printer works even after you turn off your computer. This is great for people like me who have non-standard printers. I kept forgetting to change the baud rate in RSDOS, (3 POKE instructions, I finally taped them to my monitor!), which wasted paper and drove me crazy.

Once I got FLEX I discovered that I would never have to POKE again, I'd just have to change the standard printer driver, ONCE!

A printer driver is a machine language program that hooks itself into the printer output routines of another program, such as the FLEX operating system, insuring that characters sent to the printer are usable by the printer.

The standard FLEX printer driver is named PRINT.SYS. Each computer/printer configuration running FLEX requires it's own PRINT.SYS driver. The code in PRINT.SYS is loaded into standard memory locations, and any time a character is sent to the printer, it is routed thru the PRINT.SYS code.

Those of you who have a non-Tandy printer will probably need to modify the PRINT.SYS driver. The driver supplied with CoCo FLEX is designed for printers that are connected to the CoCo thru the SERIAL I/O plug on the back of the CoCo. If you have a parallel printer, but have it connected to the CoCo with a serial to parallel converter you can use this driver. If you have a parallel printer interface that plugs into a Multi Pak Interface or a Y-Cable along with your disk controller you are going to have to work a little harder. You will either need to contact the maker of the interface to see if he has a FLEX printer driver, or get enough information to write your own.

Since I am using an OKIDATA ML 82A which has a RS-232 interface that operates at up to 1200 baud, I needed to change the PRINT.SYS driver that comes with the FLEX/CoCo package. For me the change was simple, but I am an experienced assembly language programmer. The change I describe here may not be 100% accurate for other FLEX systems, so if you have or buy another version, don't expect this to work the first time. The concepts are the same, but the details may be slightly different.

PRINT.SYS is a binary file. It is the output file created by assembling PRINTSYS.TXT, which should be included on your distribution disk. There are only two possible changes to make. The first is to change the baud rate (character transmission speed), and the second is to change the number of bits per character.

When you edit the PRINTSYS.TXT file, find the statement that contains 'ORG $CCCO'. This tells the assembler to place any machine code generated at address $CCCO. If you read your FLEX manual, you will discover that $CCCO is the standard address for any printer initialization code in FLEX. Here we will set the baud rate and bits per character for the serial output routines that come with FLEX and are used by the standard printer output routine in FLEX.

The first two instructions after the ORG should look like this:

```
LDX #$0057
STX $E216
```

There may be some comments on those lines, but they should still contain that data. The $0057 is the value that controls the print speed. This is the default for 600 baud (which is the CoCo's standard speed under RSDOS). To change that to print at 1200 baud, just change the $0057 to $0029.

The values for several common printer baud rates are:

BAUD	VALUE
110	$01CA
300	$00BE
600	$0057
1200	$0029
2400	$0012
9600	$0001 ***See Note***

Please note that I know of no printers that could be economically connected to a CoCo that can print characters fast enough to make the 9600 baud rate practical. I have seen this setting used in RSDOS with parallel printer interfaces, but the printers don't print 960 characters per second, so that setting doesn't do all that much good. This setting is not documented in the FLEX CONVERSION documentation from CPI. I have included it for those of you who are used to running your printers thru a parallel converter at 9600 baud. I DON'T KNOW IF IT WILL WORK. If you have a printer buffer this setting may help you, if it works.

The next two instructions should be:

```
LDA #8
STA $E215
```

These instructions set the number of bits per character.

The ASCII codes used by your CoCo (and all microcomputers that I know of) allow only 7 bit characters. The eigth bit is frequently ignored when it is transmitted from a computer to a modem or printer. Many modern printers allow the eigth bit to be used to generate special graphics characters. If your printer does so, leave these instructions alone. If you have an older printer, it may only accept 7 bits of data. The eigth bit will only confuse it, causing you to get garbage on your listings. If that is the case, change the #8 to a #7.

Once you have changed the PRINTSYS.TXT file you can assemble it. If you don't get any errors you can use COPY (if you have two drives) and RENAME (any configuration) to make your new driver accessible to FLEX.

For example:

```
COPY 1.0 PRINTSYS.BIN
RENAME 0.PRINT.SYS 0.PRINTSYS.OLD
RENAME 0.PRINTSYS.BIN 0.PRINT.SYS
```

The first line moves the assembler output to the system drive. If you don't have two drives, your output file is already on the system drive. The second line saves the original driver, and the third line enables the new driver.

If you have made it this far, turn on your printer and any interfaces you need, and enter the command 'P CAT'. This will send a disk directory listing to the printer. If it worked, everything is OK. If the listing is garbled, review your changes and try again.

ENOUGH IS ENOUGH

That's enough for one article. Next time I'll explain SYSTEM and WORK drive assignments, and I/O redirection (that's the I, O, and P commands). Until then, keep on computing FLEXibly.

- - -

PRESS INFORMATION

EDITORIAL CONTACTS: ED PRESTWOOD
(602) 994-6959

READER CONTACTS: MOTOROLA SEMICONDUCTOR
MARKETING HOT LINE
(800) 521-6274

MOTOROLA MICROSYSTEMS REPLACES CERTAIN DEVELOPMENT SUPPORT SYSTEMS PRODUCTS WITH RECENTLY INTRODUCED INSTRUMENTATION PRODUCTS

September 16, 1985, Phoenix AZ...Due to the wide acceptance of its more recently introduced instrumentation product lines, Motorola Microsystems has announced its intention to end production of certain items in its older 8-bit and 16-bit development support systems.

Products being terminated are:

1. EXORmacs 16-bit Development Host and Peripherals
2. EXORciser 8-bit Development Host, Peripherals, and Plug-in Expansion Boards
3. EXORset 8-bit Development Host and Peripherals
4. VMC 68/2 Microcomputer System and Peripherals
5. Certain models of Hardware Development Stations, Bus State Analyzer Personality Modules, Evaluation Modules and System Analyzers
6. All associated software products for the above lines.

Customers wishing to place final orders for these discontinued products should consult their local Motorola Semiconductor sales office for details on the exact part numbers being terminated.

Final orders for discontinued products will be accepted until 31 December 1986, with deliveries required by 31 March 1988.

###

MOTOROLA
Semiconductor Products Inc.
P.O. BOX 20912 PHOENIX, ARIZONA 85036

Innovative systems through silicon.

CoCo User Notes

by Carl Mann
30 Warren Ave.
Amesbury, MA 01913

CoCo Disk Systems, Part One
or
What Puts the "In" in "Spin"

It had to happen sooner or later. Everything I intended to write up in this month's column went wrong! The SOFTWARE has got BUGS. The HARDWARE has got BUGS. The WETWARE even has BUGS. It's the summer heat in which I'm writing. Little BUGS are actually hatching out of the office carpet!

Oh, well... By the time I see this in print, it'll all be over. In the meantime, here's Plan B. I've been saving it back for just such a time as this- NOT because it's an inferior product... but because it's so HOT a topic, I had to let it sizzle for a good while before presenting it.

Friends, the subject today is The Disk Drive. Not just Any Disk Drive, mind you... but rather a SPECIFIC Disk Drive. Maybe (even) a Disk Drive for YOUR CoCo!
Of course, there are plenty of reasons to stay away from disk drives for CoCo. Reasons like money...and the joy of listening to endless hours of 1500 baud program tapes... and money...and the straightforward qualities that only a cassette possesses... and money... and money...

Let's take a good look at the money angle. Maybe there exists more than the price of the disk system. Maybe there's another side to the question altogether: to wit, the price of your TIME. Let's face it, if the price of our time were of no importance, we'd ALL be tying knots in string to count our cattle. (That's just the tip of the cultural iceberg.) Don't misunderstand me: there are MANY times when slow is definitely superior. I think you'll agree with me that the time required for a program to load, or for CoCo to search through 90 minutes worth of tape for three or four bytes of critical data AIN'T one of 'em! Even if one's only use for CoCo is to play games (an important activity in some situations, I've learned... yes, friends, I may be gaining in softness what I lack in foolishness!)

Oh, yes. Disk systems for CoCo. As easily done as said, actually. But (for the sake of us folks that don't chase flying bit-buckets for fun, and wouldn't wish to be intently quizzed by a rabid tech-weenie on the finer points of peripheral access overhead time in relation to overall mapped register concatenation coordinates) let's back up a little. What goes on in a disk-based CoCo, anyway?

Not much different, at least on the surface. The biggest difference is that whereas the cassette tape moves in slow motion (1 and 7/8 inches per second) and only moves from left to right, the disk goes around and around at the speed of 300 RPM. Now let's see... floppy disks are 5 1/4 inches in diameter. That times pi is a circumference of about 16 1/2 inches. Let's back that off to 15 inches in deference to the spirit of American Conservatism. (Also, and more importantly, because doing so is closer to the truth. You'll see why in a minute.) Now, 300 times 15, divided by 60 seconds in a minute... why, that's exactly 75 inches per second! That's (75 divided by 1.875) exactly 40 times faster than cassette. Pretty good, no?

What this means is that the BASIC program that took 40 seconds to load from tape will be up and running from disk in about a second or so. A machine-language routine that loaded from cassette in ten seconds takes less than 1/2 second from disk. And another thing: the data may not be on the cassette, but CoCo doesn't know that. Misspell a filename, or forget which tape is which, and CoCo will stare at you as long as you care to stare back. With disk, all the files are organized in a "directory", like in a little telephone book. Each directory entry (which is automatically generated by the Disk Controller Module) tells CoCo (and you, if you care to learn to read it) exactly where on the disk to look for each file (read "thing") that CoCo put on the disk. (What YOU put on the disk may be different. Coca-Cola and cigarette ashes do not a happy disk make.) To see what is on a disk (I hope you and CoCo always agree on this point) you just type DIR <enter> from BASIC. CoCo does the rest in nothing flat.

In fact, adding a disk system to CoCo adds about 35 new keywords to Extended Color BASIC. (Unfortunately, you MUST have Extended BASIC before Disk Basic will work.) These new keywords enable the savvy programmer to do literally anything that could be imagined with the information stored (or to be stored) on the disk. By the way, that's about 250K per side of the disk. We'll look at

that later, too.) Let me tel you a secret. I have juet begun to explore what all thoee keywords can do for me on my own, and I've had disk for more than two years. Why so long a wait? It comes back to time. There is so much EXCELLENT disk software out there that I prefer to let a professional programmer sell me a "canned" program with which to do my job. I can alweye diddle with it afterwards (and almost always do) to add my personal touch of convenience, finesse, or ego gratification. (That's another thing about disk. You can add your own name and such in the appropriate placee of a commercial program- even a machine-language program- once you learn to use a "disk zapper" package. More on thoss later, too.)

Would you like to learn more about exactly how a disk system works? I could fill up an awful lot of space explaining it all, but (for the sake of the unintereeted) I would rather point to The Source Of It All. Those of us who ARE intereated are invited to write or call:

Percom Data, Incorporated
11220 Pagemill Road
Dallas, Tx 75243
(214) 340-7081

Ask for the book, "Inside Personal Computer Disk Storage Systems". It's free. It's also the very best explanation of what generally goes on in there that I have ever seen. I have lent my copy to people who didn't know a watt from a megacycle from a screwdriver, and have received profuee thanks in return. Try it on for size!

Until next time,

Carl

CEDRIC

A few weeks ago I received a copy of Cedric "A Screen-Oriented Editor for FLEX". The author, Dr. M. J. Kendall asked if I would review it. I wrote a letter back with my first impressions of the editor as it then stood, and indicated that I would review it, but that I was working on an editor of my own, and I thought it might be best to get an impartial reviewer. Mike Randall wrote back that he thought I could and would be impartial and fair and that he would still like me to review it, so here goes.

Cedric, according to the manual, "is a screen-oriented text editor specifically designed for software development, rather than word-processing. That is not to say however that Cedric cannot be used to prepare text...". It is then pointed out that the manual was prepared using Cedric. The manual indicates that the design goals were speed and flexibility.

The editor is written in assembler, a fact that by itself does not guarentee speed, but in the hands of a capable programmer, assembler code will always beat compiled code. In this case, the programmer is very capable. The whole editor occupies less than 6K of memory. Mike Randall points out that he was able to edit the entire editor source code (35 pages) and still have 14,400 bytes of memory free! The text buffer holds about 165 sectors of text. Cedric has a very nice feature in that if multiple editing functions that require updating the screen are performed, any previous screen update that is in process is aborted to do the new function. This speeds up the editing process considerably, and is a feature you would quickly come to value.

The price paid for a small code size and large edit buffer is a rather austere set of features. However for editing programs most of the time and occasional text, that price might be just right for many users. Cedric is a "single mode" editor. It is always in what most editors call the "insert" mode. That is, text entered is always inserted at the cursor. If the cursor is in the middle of a line, the text to the right is pushed along to make room for the inserted text.

One clever feature of Cedric is the availability of a menu to help you learn many of the commands. The menu commands include moving to the top or bottom of the file, getting an input file, writing an output file, writing a marked block to a disk, clearing the edit buffer for editing of another file, search for a string, replace a string with another, tab to next word, up and down screen, define a macro, repeat, and tab viewing and setting. If you need the menu to see what key to use, type ESC ESC and the menu appears. The menu tells you that T will get you to the top of the file so you type T and the first part of the file appears on the screen. If you already know that you want the T, you just type ESC T and you go to the top of the file.

The menu commands are all two-key combinations, all preceded by ESC. The single key operations are all control functions, and these must be learned by reference to the manual. Each of these is described by a two letter mnemonic such as CB for cursor back, EL for erase line, etc. Though Cedric comes with a set of pre-defined keys for these functions, a configuration file allows you to redefine the keys as desired. Thus you can set up the keyboard to be much like WordStar, Stylograph, or any other editor the key assignments of which you have memorized or become used to. These keys allow cursor motions in four directions, express cursor motions to beginning and end of current line, delete character, delete word, delete line. They allow search forward and back for a string defined by means of an ESC command, search and replace, and global replace.

There are a few well chosen convenience features. One is a counter that may be zeroed, and incremented by means of control keys. There is a command to place the value of the counter at the cursor position. This might be a nice feature for sequential label generation when programming in assembler. In a subroutine called GETA, for example, you could use labels GETA1, GETA2 etc. This counter might also be used for page numbering or illustration numbers (figure nn) in text editing.

There is no limit on line length. Entering a line longer than the screen width will eventually cause the screen to move to the right, following your entry, as beginnings of lines disappear off the left of the screen.

The macro definition feature allows you to define a macro (a series of edit command keystrokes) that can be done by means of one key. There are two "permanent" macros you can define in the configuration file. There are other control functions for tabbing, marking a block of text (which appears in reverse video as you mark it) down to the character level. Portions of text may be "cut" to a "paste buffer" and then "pasted" to somewhere else in the text.

This editor contains all the features necessary to edit a program file painlessly, and many of the desirable features of larger editors. It might be worth a quick rundown of what it doesn't have. You cannot edit a file bigger than the edit buffer, which generally is not a handicap since a program that large would usually be broken into smaller modules anyway. There is no way to set a "bell column" for a beep from the terminal, and there is no automatic "wordwrap" to the next line when a certain column is exceeded. Of course this is no handicap whatever for writing programs. Cedric has no capability to format paragraphs, but of course there are several available text processors that will do that very nicely for you.

Cedric works completely as advertised. There are no apparent bugs. The global replace function works perfectly and would have to take the all time prize for being extremely fast.

Reviewed by: Ron Anderson

Editor's Note: Since Ron did this review, during our "beta" test period, there have been several improvements to this item.

First, a very nice and easy to use terminal configuration program has been added. Just type the neceaaary keys - presto it is done! Also some additonal functions have been added. All in all, a very nice package, simple but powerful.

A **SPECIAL** introductory price of only $69.95 is a limited time deal from S. E. MEDIA.

Secondly, our policy on documentations is in order. Since we strive to bring you **good, low cost software**, we cut only those corners that make sense. NEVER to compromise quality. Thus, we will be quoting a new policy, starting this month, on documentation.

1. If you want the documentation printed out, it will be an additional $25.00, added to the advertised price. Labor, paper, etc. prices gone sky-high!

2. If you can print out the documentation yourself, you save $25.00 dollars.

3. Regardless of 1. or 2. above, the documentation will always be included on the disk. It just cost you more for us to print it out, rather than you. So again, we give you a choice!

NOTE: The above applies for S.E. MEDIA license software ONLY! All others have manufacturers documentation.

DMW

- - -

SOLVE

Symbolic Object/Logic Verification & Examination
A Super OS-9 Deluxe Debugger, Assembler and Disassembler

It has been a constant search by S. E. MEDIA to find and feature solid, economical software. We spend a lot of effort "beta testing" and consulting to insure you the very best product possible, for the price. And speaking about the price, well, this particular piece of software is, as is most all S. E. MEDIA offerings, a real bargain. It should sell for at least twice the special introduction price of $69.95. With source, in assembler, $89.95. Again we are trusting you not to comprise our faith that you will not distribute copies to anyone who has not paid their fair share. If the copyrights are violated, WE WILL ALL BE LOSERS! However, I attempt to make source available for YOUR convenience. Please don't let me down on this! I have had to really "hard sell" to get authors to furnish source at a reasonable price.

SOLVE is the most complete debugger we have ever seen, and we have had several offered for merchandising. I, Peter Dibble and others have used it for over 9 months now, and it is solid! It is simple to use and complete. It is one of the finer software products we sell. I trust you OS-9 users appreciate the effort CPI, 68 Micro Journal and especially S. E. MEDIA, has gone through to bring you offerings such as SOLVE. Normally I do not get too excited about new products, but this is an exception. It has made my development efforts decline to S I M P L E!

Overview

SOLVE has a group of monitor commands that function at the lowest level. At the next level it contains a full assembler and disassemble, both of which allow symbolic operations. SOLVE allows single stepping a program, executing it in real-time with breakpoints, or simulating its operation with a full deck of conditional traps for tracking down those elusive bugs. All commands allow symbols and/or expressions for the component parameters they expect. Sub-routine nesting is tracked at all levels and levels may be viewed in real time, simulation (about 10th speed or repositioned to any level required, (default 32 levels). Register dumps are available at any point of the session, including sub-routine nesting levels and values pointed to by the 16 bit registers. See sample below.

The disassembler will disassemble any portion of the module being acted on, at any point of the session. The assembler may be invoked at any point of the module at any point of the session also, including the memory change function. Code can be expanded by this method.

Commands

Commands are in six sections:

1. Monitor commands
2. Assembler commands
3. Disassemble commands
4. Environmental commands (breakpoints, etc.)
5. Execution commands (trace, simulate, real-time, etc.)
6. Other Miscellaneous commands and functions (below)

Commands consist generally of expressions. Expressions are evaluated left to right with no precedence. Expression results are 16 bit values, either signed or unsigned depending on the interpretation desired.

Math operators are + - * /, and may be used in symbols and/or expressions.

Constants are decimal, binary or hexadecimal numbers, the PC symbol * or single (`a) or double ("HELLO) quoted characters.

```
"AB = $4142
"ab = $6162
`A  = $41
```

There are two `base modifiers`. The `!` preceding any constant or symbol causes the current program base (if any) to be added to that component of the expression.

!0 points to the base of the program module.

The symbol `<` causes the current data base to be added to that component of the expression. Two commands ! and < allow setting these bases.

<0 points to the base of the data area.

Symbols may be strings of up to 32 characters in length. And may be pre-defined (disk file), on-line defined or written in SOLVE and written out to disk.

Examples of expressions are:

```
4+6/12-*
label*12-$1011
&23-44+label/*/&7
```

```
* = current PC value
% = binary number
& = decimal number
0/$ = hexideclmal (either is ok)
```

Errors are evaluated and reported by: -> Eval err.

SOLVE includes the capability to load symbol files from disk and write others out to disk, making all aspects of a debugging session much less painful.

Commands

Please note in the example shown at end of this article the completeness of the displays from these

commands. This IS NOT a `blind` debugger!

M - Display memory, memory contents are shown in both hexidecimal and ASCII format. Non-ASCII values are shown as `.`

C - Examine and Change memory. The address and its contents are both shown in hexidecimal and ASCII. `.` indicates non-ASCII value.

Example:

```
DBG: C $e400 <cr>  *Start at this address
E400 8D . 744 <cr>  *We entered one space and a 44
E401 06 . " HELLO <cr>  *Here we insert a string
E406 65 e $ 12 34 7734 <cr>  *Here a series expression
E40A 3A : + <cr>  *Just go one more byte
E40B 53 S =$e400 <cr>  *Go to this hex address
E400 44 D <cr> *Yep it changed
```

These features alone make this a very powerful tool. The ability to expand code, insert series of expressions, etc., puts it far above most all other development tools!

+ moves to next sequential address, - moves back one, = followed by an expression, goes to the new expression address (expression may be compounded) Enter a <space> and SOLVE outputs a ? and awaits an expression. Results are truncated to 8 bits, and next address is listed on the screen. Entering " allows entering a ASCII string, terminated by a <cr>. Entering $ allows entering a series of expressions. Expressions are evaluated to 16 bits if the expression is >$FF, otherwise an 8 bit value is stored. Terminated by a <cr>. Talk about power!

F - fills memory from expression1 to expression2 with the value of expression3.

? - searches for bytes or strings of bytes of code or ASCII. Again Expression1/2/3, as above.

Example:

```
?1111 2222,12 34 "STRING<cr>
?$e400,$3214,0D<cr>
```

All addresses of exact matches are listed to the screen, 10 addresses to a line.

```
E456 23FD F5C2, etc.
```

X - Transfer memory, again expression1/2/3 as above. Bytes from exp1 through exp2 inclusive are copied to begin at exp3. SOLVE properly handles the case where exp3 lies between exp1 and exp2.

= - calculate expression. The value of exp is calculated and listed in both hexidecimal and decimal. The results is an unsigned number in the range of 0 through 65535 decimal.

Example:

```
=$12+$1101*SYMBOL
 $0010 #00016

=10+symbol
 $0015 #00021
```

(assuming symbol has value of 5 and current program base is $10)

Assembler Mode

The assembler is invoked:

A [exp] <cr>

The assembler accepts all 6809 standard mnemonics and addressing modes. Expressions/symbols may be freely used.

The following pseudops are allowed.

```
org - change assembly origin
os9 - OS9 function call
del - deletes a particular symbol
kill - deletes all symbol table entries
equ - sets program label value
fn - sat OS9 function call value
is - sets literal label (cr is $d)
var - sets data variable.
read - reads a disk symbol file
writ - writes a disk symbol file (all current symbols)
```

Example:

```
DBG:A $e400 <cr>
E400 start leax 1,x <cr>
E402  nop <cr>
E403  bsr $e410 <cr>
E407  bra start <cr>

DBG:
```

By the interleaving of assembler and disassembler modes, practically any program can be modified at a later date with understandable labels and symbols, provided the symbol table was previously saved. Beginning to see a portion of the power of SOLVE?

P print out symbol table. The label names, values and types are shown, in that order. Types are:

```
I = immediate
O = OS9 function call
P = program label
D = data label
```

Example:

```
symbol1 7734 P
table 0001 D
f$exit 0006 O
mask 007f I
```

The assembler looks and works just like the regular assembler you are probably used to. No funnies here. Just type it in, exit with a <cr> and your code is immediately assembled to the address specified, external or internal to your program. Error checking lets you know immediately if you miscued.

Disassembler Mode

This mode disassembles memory from 6809 object code to 6809 assembly source. If you have defined labels or symbols, they will be inserted at the proper places. Also a very nice feature of the disassembler is that for branches, the destination is shown by address - no taking off the shoes and socks to do some finger and toe counting here. All joking aside, this is a real great feature for PIC code disassembly.

The disassembler is very intelligent. For example note the small portion below:

```
flag is 4
value equ $7734
 org $e400
 lda 4
 lbra $7734

disassembles as:

E400 lda $0004
E403 lbra value
```

The value `4` is not converted to `flag` since the `is` declaration makes it useable only in immediate-mode instructions. PCR instructions are shown with both the offset and the effective address. Unknown opcodes are shown with ????.

Programs or modules in OS9 that you have no source for may be linked to SOLVE and by the use of the disassembler, the Change function and assembler, completely customized new programs are easily generated, saved and `verified`, then used as any other program or module.

I don't know of any other one piece of software available for OS9 that does so much, so easily.

H - history function. SOLVE keeps a history trail (track) of the last 32 instructions executed as part of the `T` trace mode or the `S` simulate commands. It is listed as a disassembly listing.

V - define memory variables. The V command allows the contents of one or two 16 bit locations in memory to be displayed with the register dump feature. Useful for tracking the changes in program pointers or counters.

: - examine/change the User stack. Valid registers are:

A B D CC DP X Y U S PC or PCR

See sample register dumps in sample session at end of this review.

B - set & display breakpoints. Up to 16 breakpoints may be active. Memory is not altered until the `G` for go to and execute program command is invoked. Breakpoints, one, two, etc., or all can be removed with the `K` command.

^ - print stack contents. Bytes are displayed two to a line.

@ - change current nest level. This allows bypassing long loops or portions of code accessed by jump or branch to sub-routine instructions. A time saving feature for portions of code not needing debugging (hopefully).

N - set maximum nest tracking level. Any value $0 to $FF is acceptable. However, for nest levels up to $20 SOLVE keeps a special table to avoid the situation encountered by bypassing `leas` type instructions. In case of lose of nesting level by SOLVE, the `@` command can straighten things out most times.

L - link to OS9 module. Performs a F$LINK request on the module indicated. Then SOLVE branches to the `C` command, at the address of the module header sync bytes ($87CD). The program base value `!` is set to the start of the module header. The modules link count is increased by one. SOLVE stack contents are NOT altered.

E - prepare module for execution. First linking to the module listed. Then extracts the execution offset from the header and RAM requirements. The program base constant `!` is set to the start of the module header. Additional memory, if needed, is requested by SOLVE at this point. A stack is then set up for program execution. Stack can be manually set if desired, this is just a convenience.

T - Trace program (single-step). Begins execution at indicated exp. The program is executed one instruction at a time, then the registers are dumped (and some other information), pressing any key except the <cr> or keyboard abort or interrupt causes the next instruction to execute. Nesting information is updated and reported at each register dump. If the 16 nesting level is exceeded, then the program goes to real-time until the level falls below 16. If the limit is set to `0`, then only the outer most level is traced.

G - Go to program. This start the real-time execution of the program, at the PC value in the user stack, or at `exp` if given. Breakpoints are enabled for the G command only. Breakpoints can be freely moved, added or deleted, between G commands. G commands can be intermixed with S and T commands.

A special feature is that any time the program is running under the G command, a tap of any key will dump a `snapshot` of the user stack, but the program is not stopped.

S - Simulate program (multi-step). This is one of SOLVE's most powerful features. It allows a program to execute at about 10% of normal speed. Traps are enabled by several user-changable conditions.

Simulation uses the current user stack values. Breakpoints are disabled.

Simulation may be run in one of four different ways.

1. Starts at `exp`, which can be a symbol or address.

2. S alone simulates at the previous defined `exp`.

3. Conditional exit conditions are set up for simulation. Example:

S [exp] ; condition [condition....condition] <cr>

4. Deletes all conditional expressions and simulates at the previous defined count. Exp is a 16 bit value.

SOLVE executes `count` number of instructions or until a conditional becomes TRUE, whichever comes first. It then displays the condition (if one) that caused the stop, and the number of instructions actually completed. After the number is a disassembly of the next instruction to be executed, as well as a current register dump. Pressing any key except <cr> starts the next series of instructions to be executed.

There are nine types of simulation traps that can be defined. Also SOLVE traps on two additional conditions: nest level underflow and any OS9 F$EXIT service request. All traps can be used together in any combination.

The traps are listed below. The character shown on the disassembly line to show the condition that caused the stop is the same as below that defines the condition. For `F` the character is `I` or `X`.

1. F <I> <exp1> <exp2> (or)
   F <X> <exp1> <exp2>

   Frame check:

   Stop if PC is outside exp1 to exp2 (for I)
   Stop if PC is within exp1 to exp2 (for X)

   Only one I or X may be declared - not both.

2. @
   Stop if current nest level is left.

3. N <exp>
   Stop if <exp> nest level is exceeded.

4. <reg> <exp>
   Stop if stack register `reg` has value `exp`.

5. M <exp1> <exp2>
   Stop if byte at exp1 has value of exp2.

6. W <exp1> <exp2>
   Stop if 16 bit contents at address exp1 and exp1+1 has value of exp2.

7. L <exp1> <exp2>
   Stops if program passes through loops address exp1 the number of times specified in exp2, a 16 bit quantity.

8. R <exp>
   Stops on any reference to address exp.

9. O Stop after any OS9 function call.

Example:

```
S1100;x=7734,ØK,4b78,L,4b00=&100 <cr>
S ; <cr>
S 100; <cr>
S;W $e400=0832,PC=start+10 n 8 <cr>
```

$ - pass command to OS9

q - Quits SOLVE and returns control to OS9. If the module link count is one, it is removed from the module directory.

! - set program base.

```
! <exp> <cr>
```

Set the base to exp.

< - set the data base. (see above)

Either of these may be employed to set general base pointers during a debugging session with SOLVE.

So, there you have it. In my opinion one of the most valuable programming tools for OS9. Simple to use and very informative in it's display, SOLVE should be in every programmers tool kit. The power of SOLVE is far in excess of what I have been able to tell you here. I am probably one of the oldest OS9 users around. Yet, I can honestly say that it is by far the most valuable, of any programming tools I use.

SOLVE is now available from S.E. MEDIA for only:

Special Introductory Price - $69.95
w/source $89.95
(limited time offer)

DMW

- - -

Appendix B — Sample SOLVE Session

The following will not exactly match an actual SOLVE display, due to page breaks and blank lines inserted, but the match is close. Operator entries are shown in italics.

```
L.E.E. Symbolic Debug V2.3 (C)1984

DBG: r
CC A  B  DP    X         Y         U         PC   SP   -NEST-
00 00 00 0B 0000[23] 1000[E5] 0800[2F] 0000 0FFF (00/10)

DBG: f $000,$ff0 0

DBG: m $000,$b1c
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

DBG: $load name_bin

DBG: v name
CC A  B  DP    X         Y         U         PC   SP   -NEST-
00 00 01 0B 0FFF[00] 1000[E5] 0800[00] 9528 0FFF (00/10)

DBG: d
9528  LEAY $00,U
952B  LDA ,Y+
952D  CMPA $000
952F  BEQ $95A1
9531  CMPB $$2C
9533  BEQ $953D
9535  CMPA $$20
9537  BEQ $9530

DBG: s
9537  read '0,name_sym
9537

DBG: P
DRIVE.SPEC 0000 D
NAMEPTR 0010 D
OPATH 0012 D
BUFFER 0013 D
BADSPEC 9511 P
START 9528 P
DRIV.CHARS 952B P
GETNAME 9530 P
GOTNAME 9547 P
MOVENAME 956E P
NAMEND 9579 P
QUIT 9590 P
INVALID 95A1 P
ERROR 95A0 P
DBG: a 1e400

E400 cr lf $d
E400 cobee ls $2c
E400

DBG: m +0 +2f
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
9500  87 CD 00 0A 00 00 11 81 92 00 28 02 02 6E 61 6D ...........(..nam
9510  E5 49 6D 70 72 6F 70 65 72 20 73 70 65 63 69 66 .Improper specif
9520  69 63 61 74 69 6F 6E 00 31 C8 00 A6 80 81 00 27 ication.1......'

DBG: d 9528
9528 START LEAY DRIVE.SPEC,U
952B DRIV.CHARS LDA ,Y+
952D  CMPA #CR
952F  BEQ INVALID
9531  CMPA #COMMA
9533  BEQ GETNAME
9535  CMPA #$20
9537  BEQ GETNAME

DBG: r
CC A  B  DP    X         Y         U         PC   SP   -NEST-
00 00 01 0B 0FFF[00] 1000[E5] 0800[00] 9528 0FFF (00/10)

DBG: t 9528

9528 START LEAY DRIVE.SPEC,U
CC A  B  DP    X         Y         U         PC   SP   -NEST-
80 00 01 0B 0FFF[00] 0800[00] 0800[00] 952B 0FFF (00/10)

952B DRIV.CHARS LDA ,X+
CC A  B  DP    X         Y         U         PC   SP   -NEST-
80 0D 01 0B 1000[E5] 0800[00] 0800[00] 952D 0FFF (00/10)

952D  CMPA #CR
CC A  B  DP    X         Y         U         PC   SP   -NEST-
84 0D 01 0B 1000[E5] 0800[00] 0800[00] 952F 0FFF (00/10)

952F  BEQ INVALID
CC A  B  DP    X         Y         U         PC   SP   -NEST-
84 0D 01 0B 1000[E5] 0800[00] 0800[00] 95A1 0FFF (00/10)

95A1 INVALID LEAX $FFAC(BADSPEC),PCR


DBG: t name /d2 newname
CC A  B  DP    X         Y         U         PC   SP   -NEST-
00 00 0C 0B 0FF4[2F] 1000[E5] 0800[00] 9528 0FF4 (00/10)


DBG: m $0ff0 $fff
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF

0FF0  0B 00 95 28 2F 64 32 20 6E 65 77 6E 61 6D 65 0D ...(/d2 newname.

DBG: s 10;r=0ff8

00010 9537 BEQ GETNAME
CC A  B  DP    X         Y         U         PC   SP   -NEST-
80 64 0C 0B 0FF6[32] 0801[00] 0800[2F] 9537 0FF4 (00/10)

00010 9533 BEQ GETNAME
CC A  B  DP    X         Y         U         PC   SP   -NEST-
89 20 0C 0B 0FFB[6E] 0803[00] 0800[2F] 9533 0FF4 (00/10)

R-00004 953F CMPA #$20
CC A  B  DP    X         Y         U         PC   SP   -NEST-
80 6E 0C 0B 0FFB[6E] 0803[00] 0800[2F] 953F 0FF4 (00/10)

R-00001 9541 BNE GOTNAME
CC A  B  DP    X         Y         U         PC   SP   -NEST-
80 6E 0C 0B 0FFB[6E] 0803[00] 0800[2F] 9541 0FF4 (00/10)

DBG: b namend
9579

DBG: g
CC A  B  DP    X         Y         U         PC   SP   -NEST-
89 00 19 0B 0B34[72] 1000[E5] 0800[2F] 9579 0FF4 (00/10)

DBG: $ 1000;

* Exit->OS9 *

DBG: m <0 <buffer+$
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0800  2F 64 32 00 00 00 00 00 00 00 00 00 00 00 00 00 /d2.............
0810  0F FB 03 00 02 C8 17 00 59 00 01 00 00 02 00 00 ........Y.......

DBG: =badspec
$9511 #38161

DBG: c 9511
9511 49 I " A Mistake''
951C 65 e ( 2020 '0
951F 66 f  70d
9520 69 i

DBG: r

DBG: ?10 error+$10 10 3f
9533 95A1 958C 9598 959E 95A8 9582



DBG: a quit
9590  incb
959E  bra quit

95A0  nop
95A1

DBG: rPC quit y=$fff
CC A  B  DP    X         Y         U         PC   SP   -NEST-
80 03 00 0B 0B13[00] 0007[FF] 0800[2F] 9590 0FF4 (00/10)

DBG: s 1000;l quit+$10

L-00014 959D INCB
CC A  B  DP    X         Y         U         PC   SP   -NEST-
80 03 0A 0B 0B13[00] 0007[FF] 0800[2F] 959D 0FF4 (00/10)

DBG: q

OS9:
```

4490 Yukon Ct. #2A
Wheatridge, CO 80033
October 2, 1985

Dear Mr. Williams,

I have been searching (with no success) for the address and subscription cost for DTACK Grounded. Would you or any of your readers have this information?

Thanks for your time.

Sincerely,

Calvin Dodge

Calvin Dodge

STAR-KITS
SOFTWARE SYSTEMS CORPORATION

Several customers have recently asked, "What makes STAR-DOS different from FLEX (a trademark of Technical Systems Consultants Inc.), or OS-9 (a trademark of Microware Inc.)?"

STAR-DOS is most similar to FLEX. Any program which runs with FLEX also runs with STAR-DOS, but better. At this time, we know of no programs which run with FLEX but do not run with STAR-DOS.

But there are also significant differences between STAR-DOS and FLEX. Here is a short summary:

1. STAR-DOS comes with many support programs which cost extra on other disk operating systems. There are, of course, the standard ones to let you load and save files, rename or delete programs, get a disk catalog etc. But there are also others which let you list file addresses, change prompts, change or update dates, search files for text or binary data, sort disk directories by name or date, do selective copying or cataloging, recover damaged files, re-execute previously loaded programs, change the step rate of disk drives, and more.

2. For those systems which have a clock/calendar chip, STAR-DOS comes with the source code to add two extra functions. When STAR-DOS is initially booted, it will automatically get its date from the calendar chip. Furthermore, each time a file is saved to the disk, or a random file is updated, STAR-DOS will put the current date and time in the directory. This feature is important in several ways. First of all, it enables you to differentiate between similar files having the same date - you can tell what order they were saved in. We also provide a sorted catalog program called TCAT which shows the disk directory with the latest files on top. The most recent files you worked on will always be shown first in the directory listing. This ability is of tremendous help, especially to users of large disks, because it helps you find the latest files on the disk, and keeps older files from cluttering up the directory listing. Everything needed to do this is included with STAR-DOS.

3. For those systems which have extra memory, STAR-DOS comes with the source code to add either a RAM disk or disk cache. For example, with extra memory (cheap at today's prices) you can have 960K of RAM disk which will provide lightning-fast response - extended Basic loads and executes in way under one second. Everything needed to do this is included with STAR-DOS.

4. For really large systems, STAR-DOS allows you to have up to ten drives, although this does require that the disk controller(s) be capable of more than four drives, and that the disk drivers also be able to handle more drives. But even without extra drives, this means that you can have four real drives plus a RAM disk. Simply number the RAM disk as drive 4!

5. We provide other utilities for the serious user. For example, one program provided prints the drive, track, and sector numbers each time STAR-DOS accesses any disk. This provides a running record of what is happening at all times. Invaluable for the person developing new programs.

6. In some areas, STAR-DOS is substantially faster than FLEX. For example, loading time from Winchester or RAM disks is much faster (load time from floppy disks is not effected because the disk speed is the limiting factor there). Some random file operations are also much faster. For example, a test program we ran in Extended Basic to read 60 elements out of a 4000-element virtual array took 46 seconds with FLEX (with constant disk head movement), and only 15 seconds with STAR-DOS.

7. Much more significant, we have gone to extremes to make sure that STAR-DOS does not make errors. Here are just a few examples:

a. If you accidentally switch disks while a file is open for writing on the disk, any disk operating system (including STAR-DOS and FLEX) will clobber the disk. But STAR-DOS has a unique feature - it prints an error message telling you what has happened and gives you a chance to recover before further damage is done.

b. If you switch disks on a drive while there is a file open on any other drive (even though there is no file open on the disk being switched), FLEX will clobber the disk. STAR-DOS will not!

c. When a random file program exceeds the size of the random file map, FLEX will clobber the disk. STAR-DOS will not!

d. When an old random file is updated, STAR-DOS changes its date and time to show that it has been modified. FLEX keeps the old date.

e. When a program is loaded from a disk, or when the 'print error' routine is called to report an error, FLEX erases parts of the current user file control block. STAR-DOS does not!

f. When you try to access a random record which is beyond the end of an existing random file, FLEX simply returns an error message. STAR-DOS actually gives you the option of extending the file to include the desired record.

8. Perhaps most important of all, we fully support STAR-DOS and listen to our customers. When we first introduced STAR-DOS, customers reported some bugs and we fixed them. When some incompatibilities were found between STAR-DOS and existing software, we fixed them. When customers requested the ability to extend existing random files, we put it in. When a recent customer suggested that we add pipes, we added the INPIPE and OUTPIPE commands. When a customer didn't like our error messages, we added user-changeable error texts. When a foreign licensee asked for permission to translate all of STAR-DOS into French, we gladly gave it. When one young customer asked that we make the boot process a bit more friendly, we even added the word 'thank you' to the signon message.

In short, we support STAR-DOS and we support you. Even now, as we prepare our STAR-DOS for 68000 / 68008 / 68020 computers, our goal is still to make STAR-DOS as simple and user-friendly as possible. Our 68K STAR-DOS will still be totally compatible with our 6809 STAR-DOS, will be able to interchange disks with it, and will be used in much the same way. We will not abandon STAR-DOS just because we are expanding to the 68000 world!

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Road
Hixson, Tennessee 37343

Benchware Electronics
P. O. Box # 278
Holt, Michigan
48842-0278

Dear Sirs:

Benchboard BBS has been online since October 5, 1984. We operate the system 24 hours each day at 8/N/1, 300 baud. The host computer is a Tandy Color Computer with four disk drives and originates from Lansing, Michigan. The telephone number is (517) 394-2447.

Long distance callers are requested to drop a letter or post card of confirmation to the address listed above in order to access the downloads. Callers must please indicate which computer they use in order to avoid disappointment. There is a GUEST access and an online application for local callers.

Sincerely,

John C. Evans, Jr.

John Evans, SYSOP
Randy Pearson, Atari SYSOP
Ben Cranston, ML SYSOP

Dear Don,

I am another avid 68MJ reader who finds that your magazine just seems to be getting better all the time.

Perhaps I should say OUR magazine, being a Motorola micro fan. When reading your comments, I feel you're conversing with me personally.

By the way, I run a 6800 SS50 system (and badly want an MP09, used or preferably, bare board) and have recently started exploring cocoland.

Having been caught by the demise of Data Systems "68", being down the price of a couple of boards, I'd like to know if Robertson Electronics, of 1003 Warm Sands Dr., Albuquerque, NM 87123 are still selling their CLK68-1 Clock board as I haven't seen their adds for awhile.

Also on p52 of July's issue, a classified ad by a Gil Shattuck contained exactly what I want, s/h MP09 boards, clk board etc, but I hesitate because of the risk. Have you heard any complaints?

Yours Sincerely,
Steve Petschel

Steve Petschel

Editor's Note: Steve, as to the Data System boards I still hope that they will see the light of day again. I get calls, all the time, asking about these, and other kit/boards. We have that market cornered, but no one is willing to bet a few bucks on them. If we had more hands, I would be very tempted.

I believe that Robertson is still selling. We use their boards and they work fine.

As to the classifieds. There is no way we can check them out - you're on your own. However, if anyone ever got really stuck, I have not heard. I guess it has happened, but it is like buying anything else second-hand. No warranty, you takes your chances for the lesser price.

Thanks for the other nice words. Watch out for the Dingoa!

DMW

- - -

Dear Mr. Williams:

Thank you for a great magazine like 68' Micro Journal, please find enclosed a check for the amount of $66.50 in order to extend my subscription for 2 more years.

I'd like to give thanks to Dr. J. Pentecoat from G.I.T. (68MJ,Jun.82,38) for supplying me with the 6801 Tiny Basic and control related information.

Recently I had to disassemble the monitor codes (with some standard labels & comments) for the Compacta Uniboard sold by D.R.C. If some of your readers are interested in a copy, I'd be glad to send it to them.

Sincerely yours,
Prof. Geza Holzhaker
Apdo. Correos #393,
Merida 5101.
VENEZUELA. (S. America)

U.S. Postal Service Statement of Ownership, Management and Circulation (Required by 39 U.S.C. 3685): 1A. Title of Publication: 68 Micro Journal 1B. Publication no: 468510: 2. Date of Filing: 10-01-85: 3. Frequency of Issue: Monthly: 3A. No. of Issues Published Annually: 12. 3B Annual Subscription Price: $24.50 4 and 5: Complete Mailing Address of Known Office, Headquarters or General Business Offices of the Publisher: 5900 Cassandra Smith Rd., Hixson, TN. 37343 6. Full Name and Complete Mailing Address of Publisher: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343. Editor: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343., Managing Editor: Larry E. Williams, 5900 Cassandra Smith Rd., Hixson, TN 37343 7.Owner: Computer Publishing Inc., 5900 Cassandra Smith Rd., Hixson, TN 37343, whose Stockholders are: Donald M. Sr., Frances J. Williams, Larry E.Williams, Mary E.Robertson, Thomas E. Williams. 8. Known Bondholders, Mortgagees, and other Security Holders Owning or Holding 1 Percent or more of Total Amount of Bonds, Mortgages of other Securities: None 9.For Completion By Nonprofit Organizations Authorized To Mail At Special Rates: N/A 10. Extend and Nature of Circulation: A. Total No. Copies (Net Press Run):Average No. Copies Each Issue During Preceding 12 Months: 8282., Actual No. Copies Of Single Issue Published Nearest To Filing Date: 8500. B: Paid and/or Requested Circulation: 1. Sales Through Dealers And Carriers, Street Vendors And Counter Sales: Average No. Copies Each Issue During Preceding 12 Months: 4190. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 4115. 2. Mail Subscriptions: Average No. Copies Each Issue During Preceding 12 Months: 3582. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 3718. C. Total Paid Circulation: Average No. Copies Each Issue During Preceding 12 Months: 7772. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 7833. D. Free Distribution By Mail, Carrier Or Other Means Including Samples, Complimentary, And Other Free Copies: Average No. Copies Each Issue During Preceding 12 Months: 167. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 287. E. Total Distribution (Sum Of C And D): Average No. Copies Each Issue During Preceding 12 Months: 7939, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 7990. F. Copies Not Distributed: 1. Office Use, Left Over, Unaccounted, Spoiled After Printing: Average No. Copies Each Issue During Preceding 12 Months: 220, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 380. 2. Return From News Agents: Average No. Copies Each Issue During Preceding 12 Months: 133, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 0. G. Total:(Sum of E, F1 and 2-Should Equal Net Press Run Shown In A: Average No. Copies Each Issue During Preceding 12 Months: 8292, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 8500. 11. I Certify That The Statements Made By Me Above Are Correct And Complete: (Signed): MARY E. ROBERTSON, Office Manager,68 Micro Journal. Computer Publishing, Inc. 12. For completion by publishers mailing at the regular rates: Permission requested.

Gentlemen;

Here's a modified version of a patch I'm using in a bulletin board program currently under development. It might be of use to others wishing to make things easier for new or occasional users of FLEX systems.

It re-directs the error routine when the operator inputs a question mark as a command, listing a help file on the system drive called HELP.HLP instead of returning

+++ WHAT?.

HELP.TXT is the assembler input listing.
HELP.ASM is the assembled output listing.
HELP.BIN is the assembled binary output file.
HELP.HLP is a sample help text file.

Best wishes for your continued success with an excellent and most valuable publication.

Jon H. Larimore

\- - -

```
1.00=HELP.HLP
2.00=
3.00=*  Pause Control is temporarily activated.
4.00=
5.00=*  Display of this file will be shown one screen at a time.
6.00=
7.00=*  To move on to the next screen, press the ESC key.
8.00=
9.00=*  To terminate display of this file, press RETURN.
10.00=
11.00=
12.00=
13.00=
14.00=
15.00=
16.00=
17.00=
18.00=
19.00=
20.00=
21.00=
```

```
This is a standard ASCII text file, created with any of several word
processors or text editors.

It might contain:

1. Instructions for navigating a bulletin-board system.

2. Calling syntax for a word processor.

3. A catalog of other text files containing information about and
   instructions for use of specific system utilities or programs,
   along with instructions for LISTing them.

4. A catalog of printed system instruction manuals and/or part-
   numbers for specific problems.

END OF HELP FILE.
```

```
*****************************
* 6809 FLEX HELP Patch
*
* By Jon M. Laribore
* 5908 Arlington Blvd.
* Arlington, Va. 22204
*
* Last edit 08/29/85

* HELP is a patch to the FLEX DOS which will
* list a text file on the system drive called
* "HELP.HLP" whenever a "?" is input as a
* command. Instead of simply outputting
* "WHAT?".) It is intended to make the FLEX
* DOS somewhat friendlier for new or casual
* users.

* HELP will temporarily activate the TTYSET
* depth and pause controls, then return
* them to their previous settings.

* HELP normally resides in the unused FLEX
* print spooler area in a PT-69's RAM.
* If you're using this area for a spooler or
* other routine however, you may wish to ORG
* it at high memory (resetting MEMEND), or
* perhaps in another available area.

* You may either append this to your terminal
* drivers while creating a new FLEX.SYS, or
* simply GET it using the STARTUP routine.

* The "qstark" equate below is an undocumented
* location in 6809 FLEX Version 3.01. If
* you're using a different version of FLEX,
* you should verify it's accuracy prior to
* assembly.

* To verify this, use your monitor memory dump
* routine to look at address $CD09. If the
* code "8E CC56" is there, you're OK. If not,
* then again use the memory dump routine to find
* the string "WHAT?", located at $CC56 in 6809
* FLEX Version 3.01). Then, find the "8E nnnn"
* (LDX #nnnn) code in which "nnnn" is the
* address of the first byte of "WHAT?" in your
* FLEX. The address of the "8E" byte of that
* "8E nnnn" code should be your correct
* "qstark" equate.

* Also, check the value of "tradsp" here. It is
* set for a 24-line terminal display. You may
* need to change it.


* DOS Constants
*
CB8B  linbuf  equ   $cb8b     Input line buffer
C700  pspool  equ   $c700     Print spooler area (unused?)
C71B  quecnt  equ   $c71b     Print queue counter
CC03  depcnt  equ   $cc03     TTYSET display depth count
CCB9  paucon  equ   $ccb9     TTYSET pause control
CC11  lsttrm  equ   $cc11     Last terminator character
CC14  bufpnt  equ   $cc14     Input line buffer pointer
CC16  escreg  equ   $cc16     Escape return register
CD03  warmst  equ   $cd03     Warm start
CD1E  pstrng  equ   $cd1e     Print a string
CD4B  docmd   equ   $cd4b     Call DOS as a subroutine
*
CDD9  qstark  equ   $cdd9     >>> VERIFY BEFORE ASSEMBLY <<<
*
*
* If "?" is input as a DOS command, then
* list "HELP.HLP"
*
                org   qstark  Point to DOS "?" process area
CDD9 7E         fcb   $7e     Stuff "JMP" mnemonic into this byte
CDDA C71F       fdb   help1   Stuff address of this routine into
                              jump vector
* Run this routine in the unused spooler area
*
                org   pspool
*
* Delete from here to "tradsp" if you relocate
* this code to another memory area
*
* Disable print spooler jump vectors
*
C700 39 39 39   fcb   $39,$39,$39 Rts's here
C703 39 39 39   fcb   $39,$39,$39
C706 39 39 39   fcb   $39,$39,$39
C709 39 39 39   fcb   $39,$39,$39
C70C 39 39 39   fcb   $39,$39,$39
C70F 39 39 39   fcb   $39,$39,$39
*
C71B            org   quecnt  Spooler queue counter
C71B 00         fcb   0       Force Queue count to 0
*
C71C 18         tradsp fcb    24      Number of lines your terminal can display?
C71D 00         depflg fcb    0       Temporary depth flag
C71E 00         pauflg fcb    0       Temporary pause flag
*
C71F B6 CC11    help1  lda    lsttrm  Get the last non-alpha character input
C722 81 3F              cmpa   #'?     Was it a question mark?
C724 26 55              bne    help9   No, output "WHAT?"
C726 108E C78A          ldy    #dirstr Yes, point to HELP file name
C72A 8E CB8B            ldx    #linbuf Point to first character of input line buffer
C72D A6 AB      help2   lda    ,x+     Get a character, point to the next one
C72F 81 04              cmpa   #4      Is it the string terminator?
C731 27 05              beq    help3   Yes
C733 A7 80              sta    ,x+     No, store it, point to next buffer location
C735 7E C72D            jmp    help2   Store next character
C738 CC CB8B    help3   ldd    #linbuf Get input buffer address
C73B FD CC14            std    bufpnt  Store it in buffer pointer
*
* Set TTYSET terminal display lines and pause
*
C73E 7D CC03            tst    depcnt  Is terminal line depth set?
C741 26 09              bne    help4   Yes
C743 7C C71D            inc    depflg  No, set temporary flag
C746 B6 C71C            lda    tradsp  Get number of lines terminal can display
C749 B7 CC03            sta    depcnt  Store it in TTYSET depth register
C74C 7D CCB9    help4   tst    paucon  Is pause on?
C74F 26 08              bne    help5   Yes
C751 7C C71F            inc    pauflg  No, set temporary flag
C754 86 11              lda    #$11    Get "pause on" value
C756 B7 CCB9            sta    paucon  Turn terminal "pause" on
C759 CC C762    help5   ldd    #help6  Get return address for escape-return
C75C FD CC16            std    escreg  Store it
C75F BD C94D            jsr    docmd   Call DOS as a subroutine
*
* Reset TTYSET to previous settings
*
C762 7D C71D    help6   tst    depflg  Temporary depth setting?
C765 27 06              beq    help7   No
C767 7F C71D            clr    depflg  Yes, clear flag
C76A 7F CC03            clr    depcnt  And clear depth counter
C76D 7D C71E    help7   tst    pauflg  Temporary pause setting?
C770 27 06              beq    help8   No
C772 7F C71E            clr    pauflg  Yes, clear flag
C775 7F CCB9            clr    paucon  And clear pause control
C778 7E CD03    help8   jmp    warmst  Return to warm start
*
C77B 8E C795    help9   ldx    #whatst Point to "WHAT?"
C77E BD C91E            jsr    pstrng  Output it
C781 7E CD03            jmp    warmst  Return to warm start
*
C784 4C 49 53 54 0D0A   dirstr fcc    'LIST,0.HELP.HLP',$0D,4
C795 57 48 41 54        whatst fcc    'WHAT?',0
```

Canadian Concepts Limited
17 Wagoners Trail, Guelph, Ontario, Canada N1G 3M9
(519) 824 0911

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343

Dear Don:

I have thoroughly enjoyed your magazine over the years and wish you all the best in your continuing efforts to provide the dedicated users of Motorola microprocessor's with current information and editorial comments. As an engineer and microprocessor user since the days of Intel's 4004 I found that the Motorola 6800 and 6809 microprocessor designs have been easier to program, and easier to teach programming with then the Intel devices. At this time I would like to present you with my vision of a dream machine for the 68xxx users. The Apple MacIntosh was a start but no expansion and to restricted, the Commodore Amiga looks promising as well as AT&T's Unix pc, and perhaps the Pinnacle.

First the hardware requirements. The machine must use a 68000 preferably a 68020 with a 68881 math coprocessor or at least a socket and support for it. The disk controller must support DMA and handle hard disks as well as floppies. The floppies must be capable of one megabyte or as close as possible. The chassis should have the capability of mounting at least one full size five and one quarter inch hard disk, and a three inch floppy or five and one quarter inch floppy drive. The machine should support at least two serial ports as well as the console keyboard and a bidirectional parallel port that may be used as a centronics port or input from electronic devices. Four serial ports would be preferred with one of them reserved for network installation. Memory for this machine should not be limited to 512k but should start at 512k with some sockets for expansion to one megaword of memory with some buss extender for further expansion. The ideal machine should have at least medium resolution graphics in the order of 512 * 480 pixels and virtual screen support. A high resolution mode of 1024 * 1024 would be delightful. Color would be nice as well as multiple screens. Please don't default white and blue for text mode, how about black on white as the Mac or green on black with a faint border. Sprites would be an interesting way to support graphic symbols etc. as well as animation. The Amiga appears to be capable of this. The graphics outputs should include RGB digital and RGB analog as well as a composite video output. The Amiga's speach is nice as well as the sound generation chips. The system should support a mouse. The MacIntosh's mouse and the operating systems support of the mouse is a little too restrictive. I would like the mouse to have a cross hair as well as a linear motion mode for digitizing and drawing. The above requirement may require that the mouse have two motion wheels to detect rotation of the mouse. That fairly well sums up the hardware requirements except for a battery backed up time of day clock chip and that the cpu speed be as fast as possible.

The software requirements are also quite stringent. The windowing capability and pull down menu of the MacIntosh is nice for a lot of applications but having used Unix system "V" and the "csh" (c shell) with the command line expansion features and it's programming language I demand that the operating system support both. The csh has been an invaluable tool as many programs in Unix are shell executable files calling system utilities making use of various filters etc. The shell has saved the systems group a lot of programming by making use of the piping feature and various filters. The serial port drivers must be able to handle bidirectional Xoff/Xon flow control and be capable of generating and trapping a break. The ideal operating system than would probably be Os9 as in many ways it is faster and more reliable than Unix for a single user system. The machine should have some CAD/CAM software capable of generating printed wiring boards and schematics. The graphics software should support a laser printer for dumping the graphic images as well as all Hewlett Packard and Houston Instruments plotters. The programing languages should include Basic/Basic09, Pascal, C, Fortran77, STSC's APL, and perhaps ADA and Modula. These languages should all make use of the coprocessor if installed. Microware's Basic09 is fairly complete but it lacks some of the flexibility of even the Apple]['s applesoft basic. The major feature is being able to dynamically dimension array sizes at run time. This simple feature of :A=100 and Dim array(3,A) has limited Basic09 many times when trying to write universal statistics packages. It would be nice to have some data base software perhaps similar to the packages on the IBM pc's or Unix's unify.

The reasons for some of these hardware and software requirements I admit depend on the intended use. Currently I am employed at a University where we use computers to display stimuli on the crt and measure the time for a subjects response, as well as controlling laboratories, collecting data, and analysing the data. The Mac has not been a good machine as it is very difficult to measure a simple yes/no response via a parallel port or to signal external equipment other than through the serial ports, and has no multiple video screen support. Primarily we have been using the Wicat 68000 (Unix system V), Gimix 6809 system(Os9), Apple]['s, and IBM pc's (yech). The pc's are very flexible and have lots of software but I detest the architecture (ancient and slow although having math co-processor makes them faster in some applications).

Please excuse the rambling but I've been waiting patiently for this machine. The technology is here and so are the customers. Perhaps Apple has something in the works or perhaps the Amiga with Os9 may be acceptable. We as users keep hearing rumors of such machines, where are they, how much longer will we wait before buying those blue machines.

Yours Truly,
Wray Hutton

NEW PRODUCT ANNOUNCEMENT

Peripheral Technology is announcing a new single board computer, the PT-69-4. The PT-69-4 was designed specifically with OS/9 in mind, plus expanded capability. The board has 59K of user Ram as opposed to 56K in the PT-69-4. It also has two additional serial ports. The PT-69-4 features:

* 6809 Processor with 1 MHZ Clock
* Four RS-232 Serial Ports using 6850 ACIA's
* Two 8-bit Parallel Ports using 6821 PIA
* Time-of-day clock (MC146818). A battery holder is on board for a lithium battery.
* 59K of User Ram
* 4K EPROM
* 8 Double sided/double density controller chip
* Board can be configured to read standard Microware and CoCo disk formats with a single jumper change.
* OS/9 available for either the PT format or Microware CoCo format.
* Board size: 8.4" x 5.5"

PRICING: In single quantities, the board is $325. Quantity discounts are available. OS/9 for the PT-69-4 is $200.

For more information, contact:
PERIPHERAL TECHNOLOGY
1480 Terrell Mill Rd. Suite 870
Marietta, Ga 30067
404/973-0042

OS/9 is a trademark of Microware and Motorola.

Dear Don,

Many thanks for the back issues for 1985-I've been on holiday for most of August and all but the March issue have arrived already.

Please send me the 1984 issues from July to December inclusive.

Keep up the good work. I first saw microcomputers in 1976 in the Computer Store on 5th Avenue New York. They had MITS, IMSAI with the crazy front pannel stolen from a DEC PDP8, and (of course) SWTPC. I was using SWTPC gear up to a year ago (S/09 under uniFlex) til I changed jobs. I'm really pleased to see that mags and manufacturers from then are still eround.

The current machine (my first) is a 56K S50 buss box assembled by me from UK boards. Didnt build it till three years ago as I was scared of the time I knew I would spend on it-I also fly model planes competitively-so held off as long as poss. I think FLEX is ideal for this type of machine and dont see the need for more RAM. I am a professional programmer/analyst and think that huge memorys have merely made designers and implementors sloppy. My first mainframe had all of 60Kb plus 8Mb of disk. In 1970 we were running a timesharing system with 4 terminals and 6 tasks in 96Kb and 24Mb disk and it seemed fast to us then.

In general the software available under FLEX is at least as good as I have seen on many mainframes and minis. You will look in vain for a source-level symbolic dedebugger such as PL/9 has on machines like VAXen.

All the best,
M.C. Gregorie

10 Sadlere Mead
Harlow
Essex CM18 6HG
U.K.
(0279) 445174

PRESS RELEASE

CMX / SCULPTOR AT COMDEX NOV. 20-24

GIMIX will be demonstrating SCULPTOR running on the CMX 68020 UniFlex-VM system at COMDEX. COMDEX will be held from Nov. 20 through the 24th, 1985 in Las Vegas. MPD, the developers of SCULPTOR, will be occupying Booth H7171 in the new Hilton Pavillion. You are invited to stop by for a demonstration.

Classified Advertising

Tano Outpost II, 56K, 2 5"DSDD Drives, FLEX, MUMPS $895.

MICROKEY 4500 Single Board Computer, Target 128K RAM, FLEX, FORTH, with optional 6502 CPU & ROMS as advertised on p.51 Dec.84 68 Micro Journal. $2300.

LSI 68008 CPU Card with Digital Research CPM/68K $350.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, includes FLEX DOS.

TELETYPE Model 43 PRINTER - with serial (RS232) interface, and full ASCII keyboard. LIKE NEW - New cost $1295.00 - ONLY $559.00 ready to run.

S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1990. 1-DMAF2 Dual 8" Drives with Controller $2190. 1-CDS1 20 Meg Hard Disk System with Controller $2400.

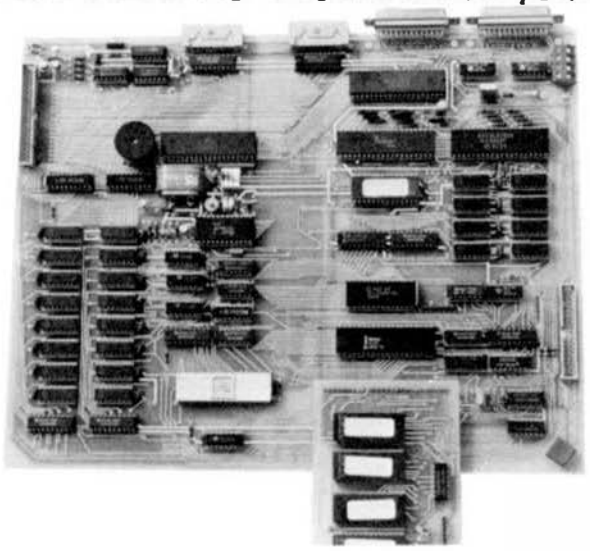
I will accept any reasonable counter-offer !!

Call Tom (615) 842-4600 M-F 9 A.M. to 5 P.M. E.S.T.

***

EXORDISC II, EXOTERM 200, EXOCISOR 784 Printer.
Tony Stein, Box 9211, Alexandria Va. 22304. (903)978-4766.

MICROBOX II Single Board Computer



Microbox II is a powerful 6809 based single board computer packed with inovative features in an easy to build form.
Running under the Flex operating system it contains 60K of dynamic ram, 8K of eprom, high resolution text and graphic displays, up to 500 sector ramdisc, up to 512 sector eprom disc, floppy disc controller, serial and parallel I/O, real-time clock and eprom programmer.
An eprom disc that looks to Flex like a standard write protected drive can be programmed with anything that would normally be on floppy - including Flex itself. A ram disc that looks like a standard unprotected disc acts as a very fast work disc. Support for two floppy drives is also on-board.
Exceptional monochrome graphic capabilities are provided by a NEC7220A graphic display controller which gives very fast drawing speeds through hardware vector, circle, rectangle, pattern and area fill generation.
The Flex operating system can be booted from any standard system disc - configuration is carried out automatically by the supplied firmware - and all the usual software can be used.
Microbox II can be controlled from a standard serial terminal a serial / parallel keyboard and video monitor or a mixture of both.

Specification:

68B09E microprocessor supporting 60K of dynamic ram and 8K firmware.
7220A graphic display controller supporting 128K of dynamic ram partitioned as monochrome video display and ramdisc.
Text display of 84x24 or 108x24 characters. Or invent your own format.
Graphic display of 768x576 pixels. Very fast hardware vectors etc.
Composite video and seperate video / sync outputs.
Eprom disc using four 27128 devices. An eprom programmer is on board.
Floppy disc controller for 48 or 96 tpi single/double density drives.
Two RS232 serial ports with programmable baudrates, 50 - 19200 baud.
Centronics type parallel printer port.
Parallel keyboard port.
Battery backed real-time clock/calendar.
DIP switch selection of input source, output destination and autoboot.
Additional I/O capability via user expansion buss.
100's of Microbox II's currently in use worldwide.

The firmware includes system diagnostics, utilities, graphic primatives, terminal emulator and auto-configuration that ensures that the board will boot from any standard Flex system disc.
The software includes disc formatter, printer drivers, disc allocation, alternative terminal emulators, eprom programmer routines, real-time clock support, graphic macros and demo, character set source and system equates.

*Flex is a trademark of Technical Systems Consultants.

Microbox II SBC bare board (12x9.5 inch) $190
plus eprom disc bare board (4x3 inch)
plus firmware and system utility disc (5.25 40 track)
Supplied with full construction and operating notes.
Firmware source code on disc. $ 29
Small "C" compiler (6809) w/7220A graphic libraries. $ 100
Small "C" compiler (6801) object. $ 130
Prices include airmail postage.

Terms: CWO. Payment by International Money Order or MASTER-CARD.

MICRO CONCEPTS 2 ST STEPHENS ROAD CHELTENHAM GLOS. ENGLAND GL51 5AA
Telephone: (0242) 510525

Introducing

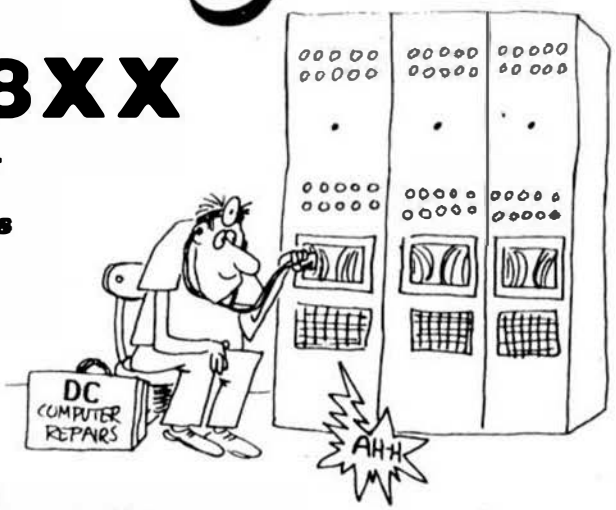
S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS



This

Not This

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and **confirm a Data-Comp service & shipping number!** This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

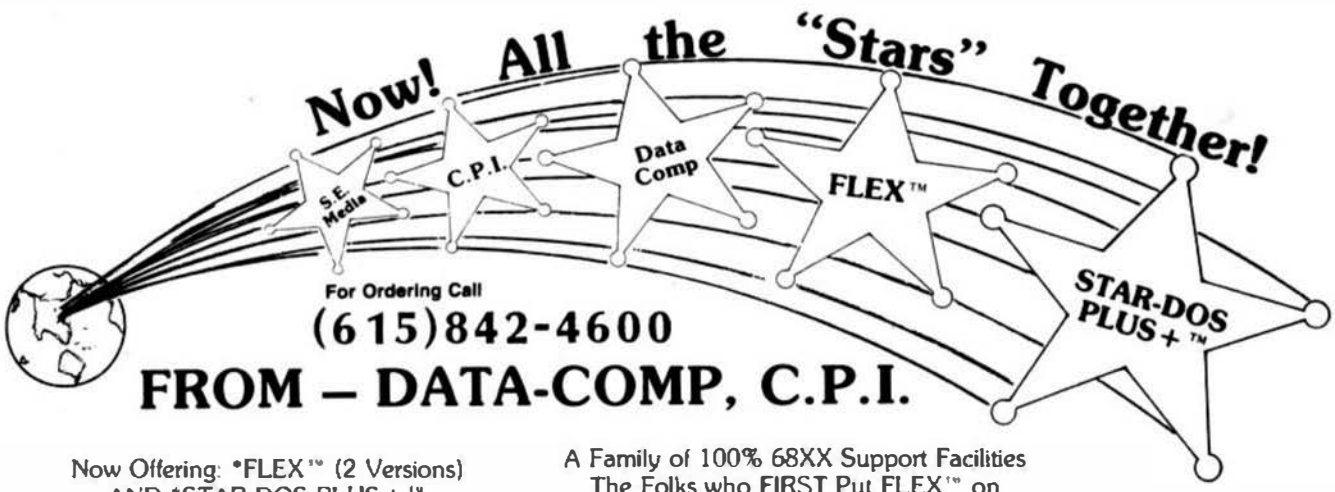
2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

☎ **(615)842-4607**
Telex 5106006630



For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+ ™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utlities
+ Super 800 Support
+ Free Color Micro Journal 1 yr. sub.
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&H NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$125.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K RXPAND TO 128K	$169.95
EPSON MX-MX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson. TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

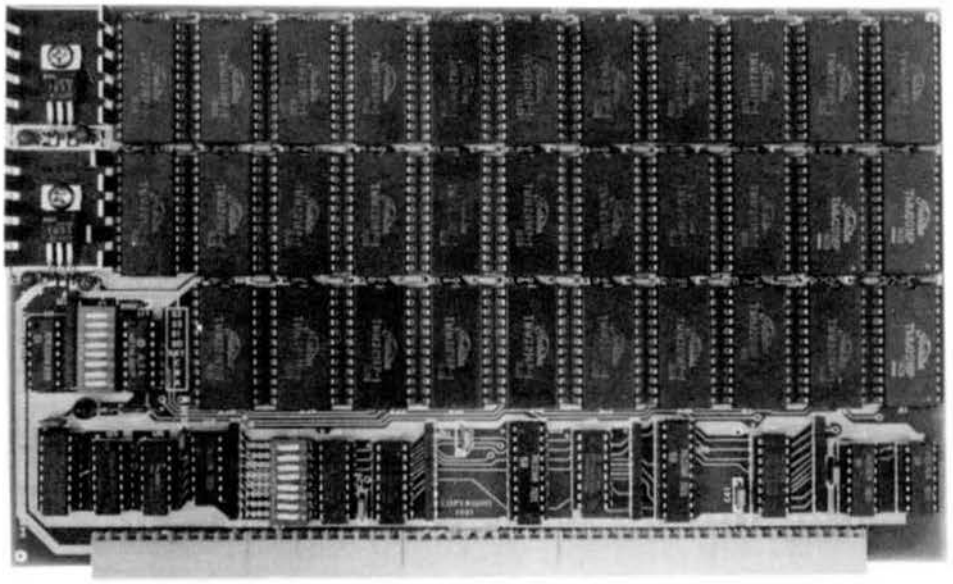
64K SS-50 STATIC RAM

PRICE CUT!!

$119.00 (48K KIT)

NEW!

LOW POWER!



RAM OR EPROM!

BLANK PC BOARD WITH DOCUMENTATION $45

SUPPORT ICs + CAPS -	$18.00
FULL SOCKET SET -	$15.00

ASSEMBLED AND TESTED **ADD $50**

FEATURES:

- ★ Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- ★ Fully supports Extended Addressing.
- ★ 64K draws only approximately 500 MA.
- ★ 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- ★ Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
- ★ 2716 EPROMs may be installed anywhere on Board.
- ★ Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
- ★ One Board supports both RAM and EPROM.
- ★ RAM supports 2MHZ operation at no extra charge!
- ★ Board may be partially populated in 16K increments.

56K	$129
64K	$139

CLOSE OUT SPECIAL
WE HAVE DROPPED OUR 32K SS-50 STATIC RAM BOARD WHICH USED 2114 LOW POWER RAMS. WE WILL SELL THE REMAINING STOCK OF BLANK PCB'S WITH DATA FOR $17.50 EA. THESE FORMERLY SOLD FOR $50.

16K STATIC RAMS?

The new 2K x 8, 24 PIN, static RAMs are the next generation of high density, high speed, low power, RAMs. Pioneered by such companies as HITACHI and TOSHIBA, and soon to be second sourced by most major U.S. manufacturers, these ultra low power parts, feature 2716 compatible pin out. Thus fully interchangeable ROM/RAM boards are at last a reality, and you get BLINDING speed and LOW power thrown in for virtually nothing.

Digital Research Computers
(OF TEXAS)
P.O. BOX 461565 · GARLAND, TEXAS 75046 · (214) 225-2309

TERMS: Add $2.00 postage. We pay balance. Order under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for insurance.

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, I2508, I2716, I2516, I2732/2732A, MCM68764/6, I2764/2764A, I2564, I27128/27128A, and I27256. (Intel, Telexas, MMotorola.

NO PERSONALITY MODULES REQUIRED!

FAI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's inteligent programming (ta) implemented for Intel 2764, 2712 and 27256 devices. Intelligent prograMming reduces the average Programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5" of flat ribbon cable for connection to the host computer MC6821 PIA interface boArd.

M68 9 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. BUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
l. RETURN ..... to the Operating system.
m. EXECUTE ..... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the Program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or licence fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (\), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAt..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESEt, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, KEEIO, FLEXIO, SCLPACK, STRSUBS, BASTRING, and REALCON.

"... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE."

Quoted from Ron Andersons FLEX User Notes column in ' 8. Need we say more?

MACE/XMACE/ASM05

All of these products feature a HIGHLY Productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- Friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also Produce ASMPROCs (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 680 /1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX V7 and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficent assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler. The TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independantly or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - Talker - Serial Poll - Single or Dual Primary Address
  - Listener - parallel Poll - Secondary Address
  - System Controller - Group Trigger - Talk only ... Listen only
- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	SS-30 interface for UPROM-II	$130.00
EXOR INT	Motorola EXORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

terms: CWO. Payment by Int'l Money Order, VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

INDUSTRIAL PASCAL FOR THE 68000

If you're looking for a language to write real-time process control software, look no further. With the rising cost of labor, it is becoming critical that a high level language be used whenever possible. Find out why over 1400 companies have switched to OmegaSoft Pascal for their demanding applications.

OmegaSoft Pascal takes the Pascal framework and expands the basic data types, operators, functions, and memory allocation to fit the needs of real-time systems. These additions fit in the same structure as Pascal and enhance its usefulness without impairing the excellent readability, ease of maintenance, and structured design.

The compiler generates assembly language for assembly and link to run on the target system. Since a true relocating assembler and linking loader is used, only those runtime modules required are automatically linked in, providing a smaller object module than other compilers.

Large Pascal programs can be split up into conveniently sized modules to speed the development process. Procedures, functions, and variables can be referenced between Pascal modules and assembly language modules by using Pascal directives.

The compiler package includes an interactive, symbolic debugger. The debugger allows setting of breakpoints, displaying and changing variables, and tracing statements. Debugging can also be done at the assembly language level when needed. The debugger allows very fast turnaround for programs to be run on the host system (target system debugger coming soon).

The compiler package also includes a full relocatable macro assembler and linking loader. These are designed to support the compiler but may also be used for general assembly language development. In addition, a full screen editor is included which can be used with a variety of intelligent terminals.

Full source code is included for the runtime library, the debugger, the screen editor, and other support utilities.

Versions to run under the OS-9/68000 and VERSAdos operating systems are currently available to end-users and OEM's. End user price is $900 (domestic) or $925 (international). A version for CP/M-68K is available for OEM use, with OEM versions for UNIX type operating systems to follow.

Similar products to run on a 6809 system and generate 6809 code are also available for most major 6809 operating systems.

Hard Disk Subsystem for SS-50 Computers

This proven subsystem adds hard disk speed and storage capacity to your computer yet requires only one SS-30 slot. Software (with source) is included for your choice of FLEX9®, OS-9® Level I or Level II, or OS-9 68K operating systems. The software honors all operating system conventions. The software is designed for the Xebec 31410 controller interfacing to any hard disk drive that conforms to the ST506 standard. Four subsystems are available:

1) 27 MB (formatted) WREN® hard disk, Xebec 31410 controller, SS-30 interface card, all cables, and software for $2850;
2) 5 MB (formatted) Shugart 604 hard disk, rest same as above for $750;
3) no hard disk, rest same as above for $600; and
4) SS-30 interface card and software for $200.

Texas residents must add sales tax. The subsystem may be mounted within your computer chassis or in a seperate enclosure with power supply. Please write (include your day and evening phone numbers) for more information. We will return North America calls so that any detailed answers will be at our expense. We are proud to announce our relocation to the growing High-Tech center of the Southwest--Austin. We regret that we will be without incoming phone service until our facilities are completed.

WELLWRITTEN™ ENTERPRISES

P.O. Box 9802 - 845
Austin, Texas 78766

FLEX is a trademark of Technical Systems Consultants, Inc.
OS-9 is a trademark of Microware and Motorola
WREN is a trademark of Control Data Corporation

OS-9 USERS GROUP

...Information Exchange
...Software Library
over 40 diskettes
...Message Of The Day
periodic newsletter

Write or Go OS9 on CompuServe for information

OS-9 Users Group
P.O. Box 7586
Des Moines, IA 50322

OS-9 is a trademark of Microware, Inc.
The Users Group is not affiliated with Microware

LLOYD I/O™

Computer Engineers

19535 NE GUSAN • PORTLAND, OR 97230 (USA)
PHONE: (503) 666-1097 • TELEX: 910 380 5448 LLOYD I O

K-BASIC™ *IS HERE*

K-BASIC is a TSC XBASIC (XPC) compatible COMPILER
for OS9 & FLEX... price $199

Here at last is a compiler for BASIC that will compile all your XBASIC programs. K-BASIC compiles TSC's XBASIC and XPC programs to machine code. K-BASIC is ready now to save you money and time by teaching your computer to:
• Think Faster • Conserve Memory • Be Friendlier

Call (503) 666-1097 for our CATALOG.
We have many programs for serious software developers!

DO™

Micro BASIC for OS9... $149

A structured mirco BASIC for general system control featuring: Parameter passing, 10 string variables, 26 numeric variables, subroutines, nested loops, interactive I/O, sequential files, and time variables (for applications executing in the background required to execute procedures such as disk or file backups.) Includes the SEARCH and RESCUE UTILITIES™. (For OS9 ONLY.)

SEARCH and RESCUE UTILITIES™

for OS9... $35

A super directory search utility. Output may be piped to the included utilities to perform file: COPIES, DELETES, MOVES, LISTING (pagination), and FILTERING. Some filtering utility programs are included; of interest is the FILE DATE CHECKING utilities: YOUNGER and DRAFT (Level 2). (For OS9 Level 1 and 2.)

PATCH™

Modem Communications for OS9... $39

PATCH is a modem communications program for OS9 featuring: KEY MACROS, ASCII TEXT AND BINARY FILE UP/DOWN LOADING, PRINTER COPY, and HELP MENUS. We use it several times each day with our TELEX service. PATCH is convenient and easy to use. Key macros may be pre-stored and loaded at any time.

CRASMB™

CROSS ASSEMBLER PACKAGE
for OS9 & FLEX... all for $399
Motorola CPU's... $150
Intel CPU's... $150, Others... $150

CRASMB is the highly acclaimed cross assembler package for OS9 and FLEX systems. It turns your 6809 computer into a development station for these target CPUs:

6800 6801 6804 6805 6809 6811 6502
7000 1802 8048 8051 8080 8085 Z8 Z80

(68000 16/32 bit cross assembler... $249)

CRASMB features: Macros, Conditionals, Long symbol names, Symbol cross reference tables, Object code in 4 formats (OS9, FLEX, S-1-S9, INTEL HEX).

VISA, MC, COD, CHECKS, ACCEPTED

USA:	LLOYD I/O (503 666 1097),	S.E. MEDIA (800 338 6800)
England:	Virocey (0582 423425),	Windrush (0892 405189)
Germany:	Zacher Computer (65 25 299),	Keil Software (04203 6741)
Australia:	Parle Radio Electronics (344 9111)	

K-BASIC, DO, SEARCH and RESCUE UTILITIES, PATCH, CRASMB and CRASMB 68.32 are trademarks of LLOYD I/O
OS9 is a ™ of Microware, FLEX is a ™ of TSC

ALL SYSTEMS INCLUDE:

- The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 5 Amps, and − 16 volts at 5 Amps.
- Gold plated bus connectors.
- Double density DMA floppy disk controllers.
- Complete hardware and software documentation.
- Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" DSDD Floppies, Cabinet & Power Supply $1698.88
20MB Streamer ..... (under development)
1.6MB Dual Speed Floppy ..... (under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only) $349.67
#64 Static RAM-64K CMOS w/battery (6809 Only) $398.64
#72 256K CMOS Static RAM w/battery $998.72
#31 16 Socket PROM/ROM/RAM Board (6809 only) $268.31

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#11 3 Port Serial-30 Pin (OS9) $488.11
#14 3 Port Serial-30 Pin (UniFLEX) $498.14
#12 Parallel-50 Pin (UniFLEX-020) $538.12
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) $618.13

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port $88.41
#43 Serial, 2 Port $128.43
#46 Serial, 8 Port (OS9/FLEX only) $318.46
#42 Parallel, 2 Port $88.42
#44 Parallel, 2 Port (Centronics pinout) $128.44
#45 Parallel, 8 Port (OS9/FLEX only) $198.45

CABLES FOR I/O BOARDS—SPECIFY BOARD

#35 Cable sets (1 needed per port) $24.95
#51 Cent. B.P. Cable for #12 & #44 ..... $34.51
#53 Cent. Cable Set $36.53

OTHER BOARDS

#66 Prototyping Board-50 Pin $56.66
#33 Prototyping Board-30 Pin $38.33
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) $545.00

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS | GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 OS9/68020	#020 UniFLEX VM
CPU Included	#05	#05	GMX III	#05	GMX III	GMX 020	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	512KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A	N/A
19MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$11,680.20	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$13,680.20	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$15,180.20	$17,180.20
GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included				
FLEX	Included	Included	Included				
GMXBUG Monitor	Included	Included	Included				
Basic 09, RunB (OS9)	Included	Included	Included				
RMS (OS9)	Included	Included	Included				
DO (OS9)	Included	Included	Included				
VDisk for FLEX	N/A	Included	Included				
RAMDisk for OS9	N/A	$125 option	Included				
O-FLEX	N/A	$250 option	Included				
Support ROM	N/A	N/A	Included				
Hardware CRC	N/A	N/A	Included				

Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

GIMIX inc.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

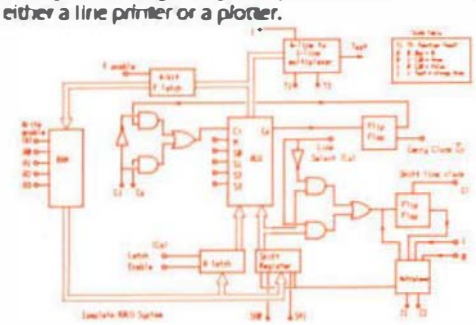
Southwest Technical Products Announces:

CAD Southwest A Computer-Aided Drafting System

Dealer Inquiries Invited

Designed for the Professional

Southwest's Computer-Aided Drafting System represents the state of the art in computer-aided drafting. It eliminates the need for labor intensive drafting using the traditional T-square and triangle approach. Via keyboard or a sophisticated digitizer tablet, you can quickly and easily prepare even the most complicated drawings. Naturally, changes can be easily made at any time. A library of standard symbols is provided as well as the flexibility of a user defined symbol library. Drawings may be printed on either a line printer or a plotter.



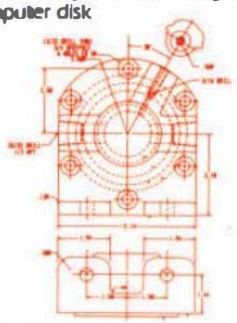
Increased Productivity

A powerful set of commands allows even the novice user the ability to quickly and easily prepare complex drawings. The experienced draftsman will quickly realize substantial improvements in productivity. The computer assumes the entire burden of line and arc drawing, freeing the user to concentrate on design issues. A partial list of key features includes:

- no limit to number of drawings
- standard grid pattern, used for reference; will increase or decrease with scale
- isometric grid pattern - similar to standard
- library mode - complete objects can be drawn and stored in users' library for later use in any drawing
- mirror objects - images of objects in library can be easily duplicated
- the terminal screen acts as a window by which the user can quickly move large drawings
- draw/delete functions:
  - arc/fillets (solid or dashed)
  - circles (solid or dashed)
  - lines and sequence of lines (solid or dashed)
  - true horizontal/vertical lines
  - center lines
  - dimension lines
  - extension lines
  - dimension lines perpendicular to extension lines
  - extension lines perpendicular to normal lines
  - boxes: any shape or size defined by two diagonal endpoints
  - library objects: previously defined library objects drawn or deleted with a couple of keystrokes
  - area fills
  - special functions:
    - change line mode: solid to dashed, dashed to solid
    - sector circle: save portion of circle as arc
    - segment lines: break single line into two lines
  - scale using factor - choose desired scale multiple
  - scale using window - "zoom in" and work on a selected area
- complete set of labels - comprehensive label generation component to annotate drawings
- overlays - any drawing can have up to 255 overlays
- cursor functions:
  - variable size: user can scale cursor to convenient size
  - variable step: user can select single pixel/giant step for movement of cursor
  - homing: cursor can be quickly moved to 9 possible predefined home positions on screen

ARCHITECTURAL DESIGN USING CAD-SOUTHWEST DRAFTING SYSTEM

- complete disk-oriented library maintains routines that allow you to maintain your drawings on the computer disk



Big System Features

As you prepare your drawing, you can control which portion is available for work on the terminal screen. Simple commands allow you to move up, down and sideways to view different segments of a large drawing. "CAD ZOOM" allows you to quickly "zoom in," turning your terminal into a computerized magnifying glass for detailed review of individual sections. "CAD ZOOM" compresses the drawing, allowing you to look at multiple sets of the drawing on one screen.

A powerful overlay feature supports up to 255 overlays for the same drawing. For example, you may wish to have the floor plan on one overlay, electrical wiring on a second, HVAC on a third, and so forth.

The Right Equipment

CAD Southwest runs on the proven SWTPc family of computers operating under UniFLEX®, a UNIX™ like operating system. Five to eighteen terminals, depending on the model number, can be attached to the CPU with a variety of hard disk storage for your drawings.

The industrial quality X-12 terminal from Southwest provides crisp resolution and a 92-key keyboard. A digitizing tablet may be optionally attached to each terminal.

Drawings may be printed on a dot matrix printer or optionally on an 8 color pen plotter.

UNIX™ is a trademark of AT&T Bell Laboratories. UniFLEX® is a registered trademark of Technical Systems Consultants, Inc.

Southwest Technical Products, Corp.

219 West Rhapsody • San Antonio, Texas 78216

(512) 344-0241

Southwest Technical Products Announces:

The X-12+. . .
The Solutions System

Dealer Inquiries Invited

The X-12+ System is a state of the art Systems Solution for multi-user data-intensive applications.

Solutions Hardware

At the heart of the system is a Motorola 68B09 processor delivered with 256KB (optionally expandable to 1 megabyte), and bench-marked at performances comparable to those of several Motorola 68000 UNIX™ systems.

The central processing unit, main memory and four RS232 serial ports are housed in the CRT cabinet enclosure.

The X-12 CRT is a professional quality unit which includes a detached 92 key keyboard including 15 function keys, cursor control keys, and a numeric keypad.

Next to the X-12 CRT sits the mass storage sub-system. The X-12+ Model WC20 includes a 22 megabyte (formatted) half height 5¼" Winchester Mass Storage Device with an 85 millisecond average access time and a 5¼" floppy disk with 1.25 megabytes. The X-12+ WC40 system includes a 33 megabyte (formatted) Winchester disk with an average 85 millisecond access time.

The WC40 Model includes a 5¼" floppy as well as a 40 megabyte streaming tape subsystem.

A variety of additional peripherals are available for X-12+ systems including printers, digitizers, and plotters. Both systems provide the user with an exceptional price/performance solution.

Solutions Software

UniFLEX® (a UNIX™ look-alike) provides a powerful multi-user, multi-tasking operating system. UniFLEX® resides in less than 60,000 bytes of main memory. Naturally, the X-12+ Solutions System supports Basic, "C", Cobol, Fortran, and Pascal, making the X-12+ an ideal development environment for a variety of applications.

Data Base, Word Processing, and Spread Sheet programs are only a few of the many application products available for the Solutions System. A variety of vertical application products are available through an international Network of Southwest Dealers.

Southwest Technical Products Corporation, provides complete manufacturing, training, maintenance and marketing support to its dealers and users from its 68,000 square foot facility in San Antonio, Texas. Its 20 year commitment to excellence is demonstrated by the X-12+ family of Systems Solutions.

UNIX™ is a trademark of AT&T Bell Laboratories. UniFLEX® is a registered trademark of Technical Systems Consultants, Inc.

Southwest Technical Products, Corp.

219 West Rhapsody • San Antonio, Texas 78216

(512) 344-0241